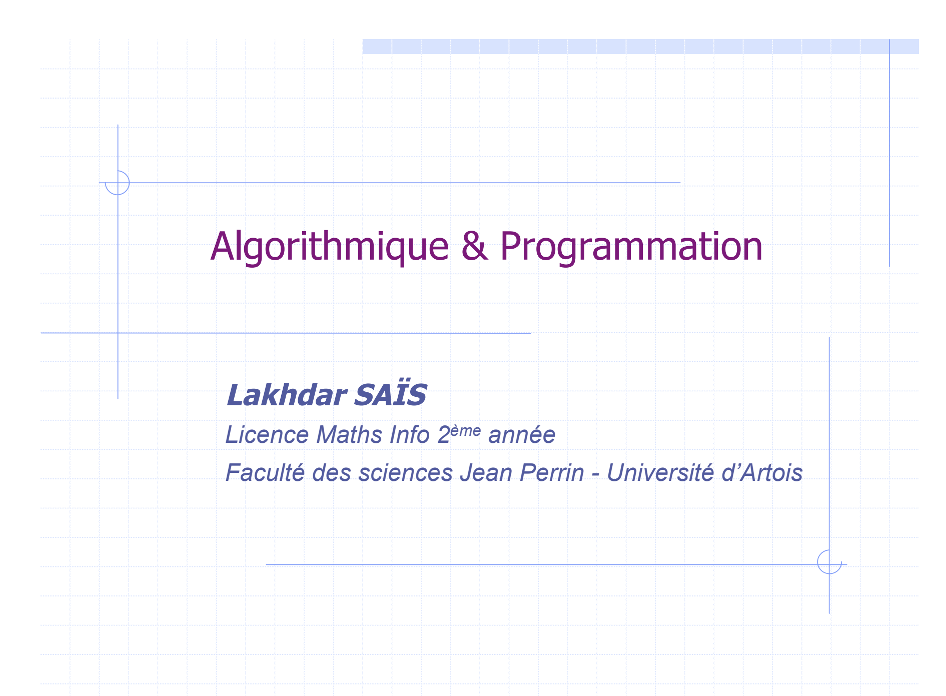


Présentation

- Cours :
  - ***Responsable*** : Lakhdar SAÏS
  - ***Contact*** :
    - **CRIL, Bureau 307**
    - **sais@cril.fr**
    - **http://www.cril.fr/~sais**
  - ***Cours*** :
    - **2 séances /sem**
      - **Lundi : 10h45 - 12h15 (Amphi S19)**
      - **Vendredi : 9h30 - 10h30 (Amphi S23)**

Présentation

- TD :
  - ***Intervenants*** :
    - Lakhdar Saïs
    - Nathalie Chetcuti
- TP :
  - ***Intervenants*** :
    - Lakhdar Saïs
    - Nathalie Chetcuti

Présentation

- Les examens :
  - ***Contrôles*** :
    - CC : la semaine du 29 octobre - 4 Novembre 2007
    - Examen : la semaine 17 -21 décembre 2007 (1ère session)
      Seconde session : semaine du 22-6 janvier 2008
    - TP : 1 projet + 1 contrôle de TP
  - ***Évaluation*** :

$$Note = \frac{3}{4} \times Sup(\frac{(Ex + Ds)}{2}, Ex) + \frac{1}{4} \times Tp$$

☞

Si (Note>=10)
alors écrire(' Admis ')
sinon écrire(' Ajourné)
écrire(' Une nouvelle chance en seconde session!!')
fin si

Plan

- Introduction
- Rappel
- Procédures & fonctions
- Récursivité
- Pointeurs
- Listes, piles, files,...
- Arbres
- Fichiers

➢ **Langage : Pascal**

Références bibliographique

Livres :
[1] Jacques Courtin, Irène Kowarski, « ***Initiation à l'algorithmique et aux structures de données*** », volume 1 et 2, Dunod
[2] Guy Chaty, Jean Vicard, « ***Programmation : cours et exercices***», Ellipses

 Utiliser internet et la bibliothèque!!

 Poser des questions???

Introduction

- Algorithme ?
  - Vous avez déjà ouvert un livre de recettes de cuisine?
  - Avez-vous déjà déchiffré un mode d'emploi traduit du coréen pour faire fonctionner un magnétoscope?
  - Si oui?
    - Sans le savoir vous avez déjà exécuté des algorithmes.

  Encore plus fort :
  - Avez-vous indiqué un chemin à un touriste égaré?
  - Avez-vous fait chercher un objet à quelqu'un par téléphone?
  - Si oui?
    - Vous avez déjà fabriqué - et fait exécuter- des algorithmes
- Définition : *Un algorithme est une suite d'instructions qui une fois exécutée correctement, conduit à un résultat donnée.*

Introduction

- Exemple 1 :
  - *Un algorithme de résolution de l'équation ax+b = 0*

données : a et b entiers

Algorithme :

Écrire(' résolution de l 'équation : ax+b=0 ')
lire(a), lire(b)
Si a est non nul,
alors on obtient la solution : x = -b/a
sinon si b est non nul,
alors l'équation est insoluble
sinon tout réel est solution

Résultat : la solution de l 'équation ax+b=0; si elle existe

Introduction

- Programme?
  - Remarque : l'algorithme de calcul du pgcd ne peut être exécuté par une machine! **(qui ne comprend que des 0 et des 1)**

    Besoin d'un langage de programmation (ici le langage Pascal)
  - Un programme est la traduction d'un algorithme dans un langage informatique particulier et parfaitement précis.

Algorithme :

Écrire(' résolution de l 'équation : ax+b=0 ')
lire(a), lire(b)
Si a est non nul,
alors on obtient la solution : x = -b/a
sinon si b est non nul,
alors l'équation est insoluble
sinon tout réel est solution

```
program equation (input, output);
var a,b: real;
begin
writeln(' Résolution de l 'équation :
 ax+b=0');
write(' a ? ');readln(a);
write(' b? '); readln (b);
if ( a <>0)
then writeln(' x =' ,-b/a)
else if (b<>0)
   then writeln(' insoluble')
   else writeln(' tout réel est solution')
end.
```

Choisir

Introduction

- Exemple 2:
  - *Calcul du PGCD de deux entiers positifs :*

données : a et b entiers positifs

Algorithme :

Appliquer récursivement les deux règles suivantes :

R1) - PGCD(a, 0) = a

R2)- PGCD(a, b) = PGCD(b, a modulo b), b≠ 0

Résultat : le pgcd entre a et b

Exécution : PGCD(36, 24)?

PGCD(36,24) =R2 PGCD(24, 12) =R2 PGCD(12, 0) =R1 12

Introduction

```
program Calcul_pgcd (input, output);
var a,b, reste: integer;
begin
  write(' a ? ');readln(a);
  write(' b? '); readln (b);
  while(b<>0) do
   begin
     reste:=a mod b;
     a:= b;
     b:=reste;
   end;
  writeln(' pgcd = ',a);
end.
```

Itérer

Introduction

- Stuctures de données ?

  Définition : une **structure de donnée** est un moyen de coder et de structurer les données manipulées.

Exemple :

type **suite** = tableau [1..N] d'entiers

etudiant = structure

nom, prénom : chaîne de caractères

âge : entier positif

sexe : caractère

...

fin structure

Données
Algorithme
Résultats
Structures de données

Programme = Algorithme + Structures de données

Introduction

Problème :

Trouver le plus court chemin entre Lille et Marseille

Données Graphe

Algorithme recherche du plus court chemin entre deux villes

Résultats Chemin

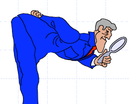
Graphe, chemin ?

- Structure de données

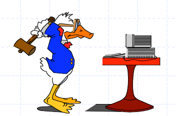
Lille
30 km
220 km
Lens
210 km
Paris
800 km
750 km
650 km
Toulouse
200 km
Nice
Marseille
200 km

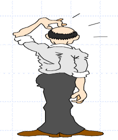
Introduction : résolution de problèmes

- **Lire l'énoncé du problème, être certain de bien le comprendre**
  - utiliser une loupe
  - ressortir les informations pertinentes
  - données ? résultats ?



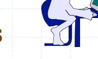
- **Réfléchir à la résolution du problème en ignorant l'existence de l'ordinateur**
  - déterminer les points principaux à traiter
  - exploiter l'ensemble de vos connaissances
  - adapter ou réutiliser des recettes existantes
  - encore difficile ?
    - Décomposer le problème!! ***Analyse descendante***!!





Introduction : résolution de problèmes

- **Écrire formellement la solution (algorithme) sur papier**
  - utiliser un pseudo langage
- **Vérifier votre solution sur un exemple**
  - preuve formelle de l 'algorithme : encore mieux !!
- **Traduire dans un langage de programmation**
- **Tester le programme sur différents jeux de tests**
  - le jeux de tests doit être « suffisant »
  - ne pas oublier les cas particuliers, …



Introduction

- **Exemple : conversion de nombre**
- **Énoncé**

  Écrire un programme qui simule la conversion d'un nombre décimale en tant que chaîne de caractères en sa valeur réelle.

  Exemples de nombres à lire : 12 45.3 0.9625 (notation décimale)

- **Analyse**
  - les nombres ne sont pas des valeurs isolées.
  - font partie d'un texte.

    Exemples :
    - « la valeur de x est 1526.32 »
    - « 12 * 5.01 + x = y »

  Le programme doit :
  - lire tous les caractères jusqu'au premier caractère chiffre ;
  - lire la partie entière du nombre,
  - et s'il trouve le point, la partie décimale.

Introduction

Exemple *:*

le mot 15.67 sera traduit en la valeur entière 1567, cette valeur sera ensuite divisé par 10^2 (2 étant le nombre de chiffres après la virgule)

- **Algorithme**
  1. rechercher un chiffre
  2. lire la partie entière du nombre
  3. ***si*** il y a un point décimal

     alors

     3.1. lire les chiffres suivants ce point

     3.2. remettre à l'échelle ce nombre

     fin si
  4. écrire le résultat

Introduction

1. rechercher un chiffre

répéter lire un caractère ***jusqu'à*** trouver un caractère chiffre.

2. lire la partie entière du nombre

répéter

2.1. convertir le caractère chiffre en sa valeur entière

2.2. rajouter cette valeur à (nombre déjà calculé)*10

2.3. lire le caractère suivant

jusqu'à ce que le caractère lu ne soit plus un caractère chiffre

Introduction

3. lire la partie décimale

si le caractère lu précédemment est un point

alors

3.1. lire le caractère suivant

tant que le caractère lu est un caractère chiffre faire

3.1.1. convertir le caractère chiffre en sa valeur entière

3.1.2. rajouter cette valeur à (nombre déjà calculé)*10

3.1.3. comptabiliser le nombre de chiffre après le point

3.1.4. lire le caractère suivant

fin tant que

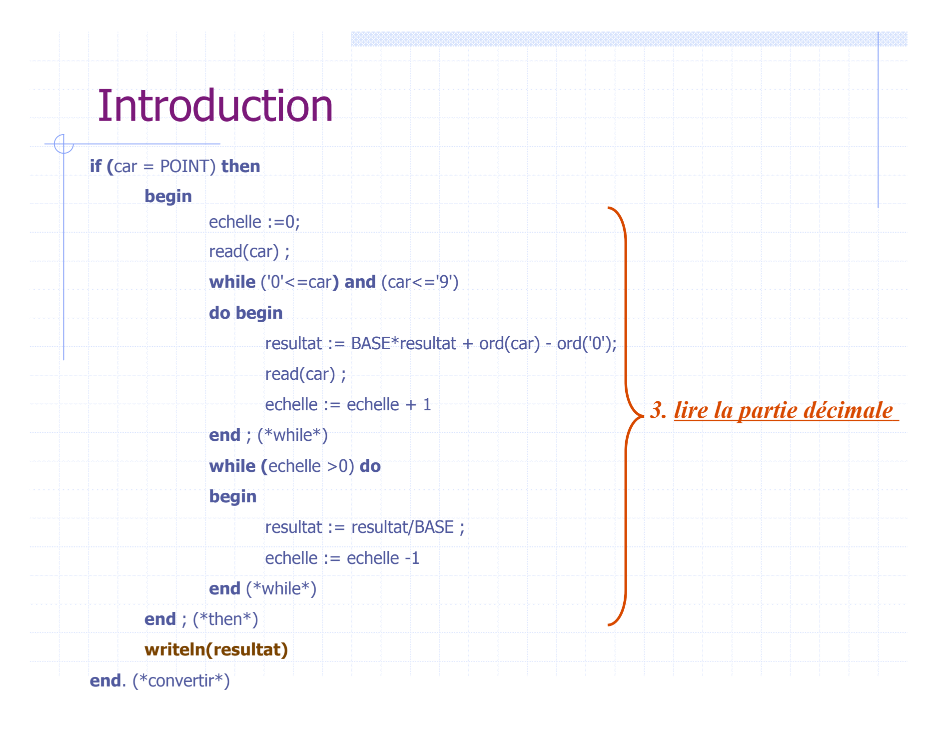
3.2. diviser le nombre obtenu par 10 puissance (le nombre de chiffres lus après la virgule)

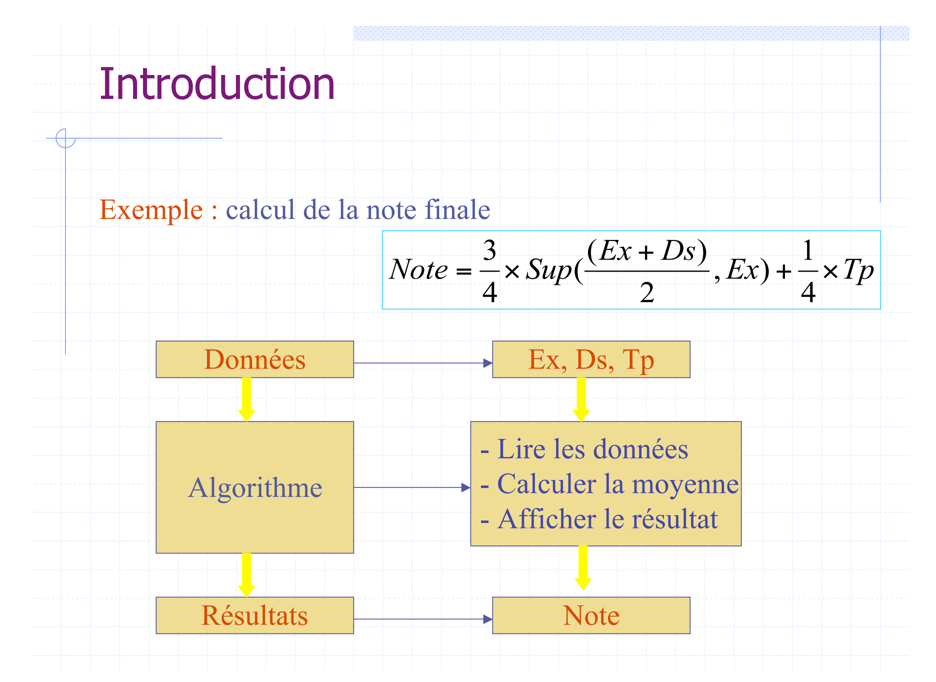
fin si

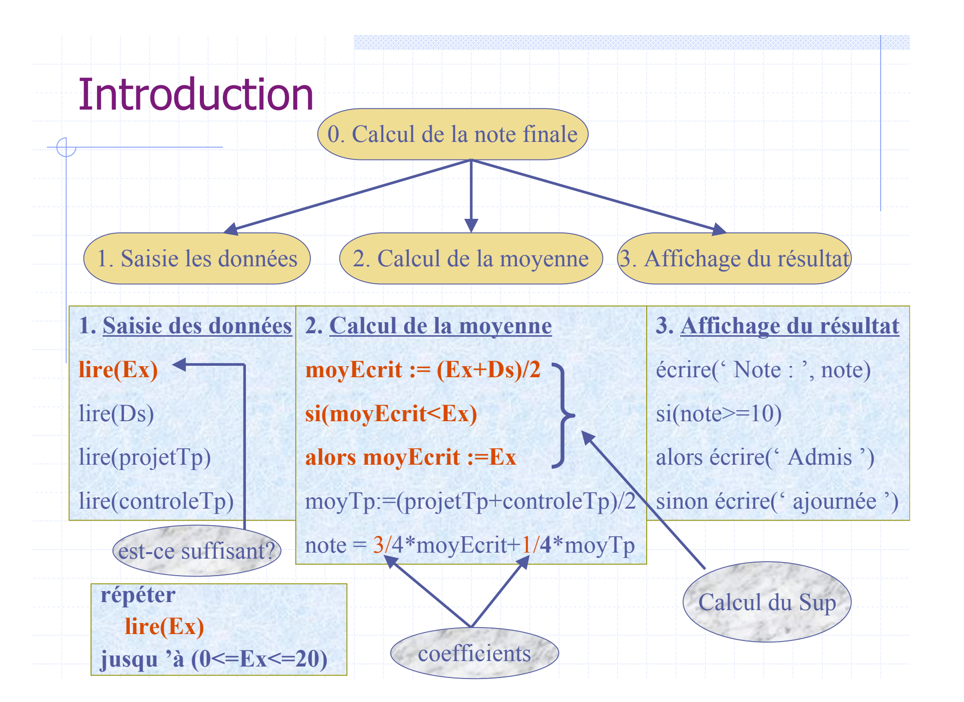
Introduction

Programme Pascal

```
Program convertir_repeat (input, output) ;
const
   POINT = '.' ;
   BASE = 10 ;
var
   resultat : real ;
   echelle : integer ;
   car : char ;
begin
   repeat read(car)
   until ('0'<=car) and (car<='9') ;          } 1. rechercher un chiffre
   resultat := 0;
   repeat
        resultat := BASE*resultat + ord(car)-ord('0') ;   } 2. lire la partie entière
        read(car)
   until ('0'>car) or (car>'9');
```










```
program notes(input, output);
   {Déclaration  des constantes et des variables}
   const COEF_ECRIT = 3; COEF_TP = 1;
         B_INF = 0; B_SUP=20;
   var ex, ds, moyEcrit : real;
         projetTp, controleTp, moyTp:real;
         note : real;
begin {Saisie des données }
repeat
  write(‘ Exam? ’);readln(ex);
until (ex>=B_INF and ex<=B_SUP);
repeat
  write(‘ Ds? ’);readln(ds);
until (ds>= B_INF and ds<= B_SUP);
repeat
  write(‘ Projet de Tp? ’);readln(projetTp);
until (projetTp>= B_INF and projetTp<= B_SUP);
repeat
  write(‘ contrôle de Tp? ’);readln(controleTp);
until (controleTp>= B_INF and controleTp<= B_SUP);

{ Calcul de la moyenne }
moyEcrit := (Ex+Ds)/2;
if (moyEcrit<Ex)
then moyEcrit :=Ex;
moyTp:=(projetTp+controleTp)/2;
note=(COEF_ECRIT*moyEcrit+COEF_TP*moyTp)
      /(COEF_ECRIT+COEF_TP);
{Affichage du résultat }
writeln(‘ Note : ’, note);
if (note>=10)
then writeln(‘ Admis ’)
else writeln(‘ Ajournée ’);
end.
```

Rappel

- **Algorithme & programme : quelques briques fondamentales**
  - l'***affectation*** de variables
  - la ***lecture/écriture***
  - les ***tests***
  - les ***boucles***

  ⇘ - un algorithme(prog) se ramène toujours à une combinaison de ces quatre petites briques.

  - ces briques ***opèrent*** sur des ***données*** (***variables, constantes***) de ***différents types***.

Rappel

Un programme manipule deux types de données :

- **Variable : une variable est un objet bien identifié dont les valeurs peuvent être altérées au cours de l'exécution du programme.**
- **Constante : une constante est un objet bien identifié dont la valeur est fixée une fois pour toute, et ne peut être modifiée au cour de l'exécution du programme.**
- **Déclaration :**

  const NOM_CONST = valeur;

  var nomVar : **type**;

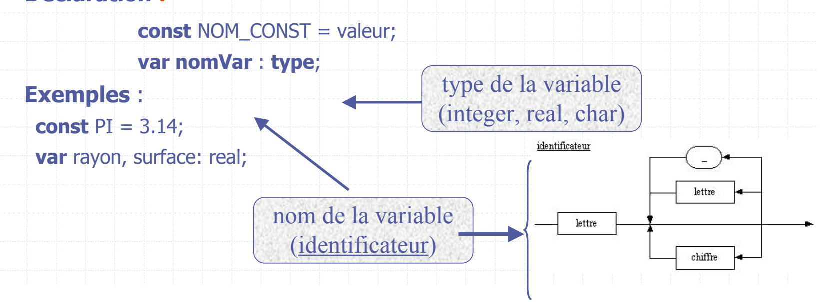
  Exemples :

  const PI = 3.14;

  var rayon, surface: real;

type de la variable (integer, real, char)

nom de la variable (identificateur)



Rappel

- **Type de données standard**

Type de données	Exemple de type
Simple	Integer, Real, Boolean, Char, type énuméré
Chaîne	String
Structuré	Array, Record, Type ensemble, Type fichier
Pointeur	Type Pointer

Rappel

- **Type entier**

Type	Intervalle de valeurs	Espace mémoire occupé
byte	[0..255]	1 octet
shortint	[-128..127]	1 octet
word	[0..65535]	2 octets
integer	[-32768..32767]	2 octets
longint	$[-2^{31}..2^{31}-1]$	4 octets
Opérations	**signification**	**Exemple**
+, -, *, /	…	9/5 =1.8
div	Division entière	9 div 5 = 1
mod	Reste de la division	9 mod 5 = 4
succ	successeur	Succ(5) = 6
pred	prédécesseur	Pred(5) = 4

Rappel

- **Type réel**

Type	Intervalle de valeurs positives	Espace mémoire occupé
real	$[2.9\ 10^{-39}..1.7\ 10^{38}]$	6 octet
single	$[1.5\ 10^{-45}..3.4\ 10^{38}]$	4 octet
double	$[5.0\ 10^{-324}..1.7\ 10^{308}]$	8 octets
extended	$[3.4\ 10^{-4932}..1.1\ 10^{4932}]$	10 octets

- **Opérateurs classiques** : +, -, *, /
- **Autres fonctions prédéfinies** :

  abs(x), sqr(x), sqrt(x), sin(x), ln(x),…trunc(x), round(x)…
- **Opérateurs de comparaisons** : =, >, >=, <=, <>

Rappel

- **Type caractère : char**
  - **chiffres** :' 0 '.. ' 9 ',
  - **lettres** :' A'.. ' Z ', ' a '.. 'z ',
  - **caractères spéciaux** : ' + ',- ', … '? ', ' * ', ' ', ' _',…
  - **les caractères sont ordonnées** selon un code, unique pour chaque caractère : le plus courant est le code ASCII.
  - **Les opérations** possibles sur les caractères sont:
    - **les comparaisons : =, <>, <=, <, >=, >**
    - **succ, et pred.**
    - **ord(c) retourne le code ASCII de c,**
    - **chr(n) retourne le caractère dont le code ASCII est n**
    - **upcase(c) retourne le caractère majuscule de c, et c sinon.**
  - Quelques soit le code utilisé, on a toujours :
    - 'A'<'B'<…<'Z'
    - 'a'<'b'<…<' z'
    - '0'<'1'<…<'9'

Rappel

- **Type Booléen : Boolean**
  - un booléen admet deux valeurs : false (Faux) et true (Vrai)
    - **Ordre** : false < true,
    - succ(false) = true, pred(true) = false
  - **Opérations :**
    - and (et), or(ou), not(non), xor (ou exclusif)
    - = , <>, <=, <, >=, >

Rappel

- **Type énuméré**

  un type énuméré permet de construire explicitement les valeurs de ce type. Nous pouvons écrire:

  type jourSemaine = (lundi, mardi, mercredi, jeudi, vendredi, samedi, dimanche);

  couleurs_arc_en_ciel = (rouge, orange, jaune, vert, bleu, indigo, violet);

  Les valeurs d'un type énuméré ordonnées selon leur ordre de déclaration :

  Pour le type jourSemaine nous avons

  lundi < mardi < mercredi < jeudi < vendredi < samedi < dimanche

  Remarques: A chaque valeur énumérée correspond un numéro d'ordre (nombre entier). Dans notre exemple lundi porte le numéro 0, mardi porte le numéro 1 …

  Les opérations possibles : = <> < <= > >=, succ, pred, ord

Rappel

- **Type intervalle**

  Un intervalle permet de restreindre le groupe des valeurs d'un type appelé type de base et choisi parmi integer, boolean, char ou un type énuméré.

  Exemples:

  - **0..9** est l'intervalle des valeurs entières 0 à 9, le type de base est integer
  - **lundi..vendredi** est l'intervalle des valeurs énumérées lundi à vendredi, le type de base est jourSemaine
  - **'A'..'Z'** est l'intervalle des lettres majuscules, le type de base est char

```
type   chiffre = 0..9;
         joursDeTravail = lundi..vendredi;
         majuscule = 'A'..'Z';
```

  Les opérations possibles sont celles du type de base.

Rappel

- Entrées/Sorties :

  Lecture : *lire* en pseudo

  - *read(suite de variables)* (resp. *readln(suite de variables)*) permet de saisir des données au clavier, le curseur reste sur la même ligne (resp. saute une ligne)

  Exemple :

  - ***read (v1, v2, ..., vn);* équivaut à *read (v1); read (v2);... read (vn);***
  - ***readln (v1, v2, ..., vn);* équivaut à *read (v1);read (v2);...read (vn); readln***

  Règle pratique :

  Pour éviter toute surprise, chaque fois qu'une donnée tapée par l'utilisateur se termine par une marque de fin de ligne, cette marque doit être sautée (éliminée) par l'utilisation de readln.

Rappel

- Entrées/sorties :

  Ecriture : *écrire* en pseudo

  - *write(expression)* (resp. *writeln(expression)*) permet d'afficher des données au clavier, le curseur reste sur la même ligne (resp. passe à la ligne suivante)
    - **expression peut être un texte, le contenu d 'une variable, résultat d 'un calcul.**
  - ***write(v1, v2, ..., vn);* équivaut à *write (v1); write (v2);... write (vn);***
  - ***writeln (v1, v2, ..., vn);* équivaut à *write (v1);write (v2);...write (vn); writeln;***

- **Exemple :**

```
var  rayon : real;     (* rayon d'un cercle *)
...
write ( 'Veuillez donner le rayon du cercle: ' );
readln ( rayon );
```

Rappel

- Affectation :

  En pseudo : variable ← expression

  En pascal : variable := expression;

  Exemple :

  moyTp:= (projetTp+controleTp)/2;

  Remarque :

  le type de la variable et de la valeur de l'expression doivent être identique (dépend du langage).

Rappel

- **Tests :**

En pseudo : si condition

```
si condition
    alors instruction_1
    sinon instruction_2;
    fin si
```

En pascal : if condition

```
if condition
    then instruction_1
    else instruction_2;
```

condition est une expression booléenne

Exemple :

```
if (nb mod 2 = 1)
then begin
        write( 'Le nombre ',nb);
        writeln(' est impair');
     end
else writeln( 'Le nombre ',nb ,' est pair');
```

Rappel

- Tests :

Questions: peut-on remplacer l'instruction conditionnel par l 'affectation?

```
var pair : boolean;
    nb: integer;
begin
  …
if (nb mod 2 =0)
then pair :=true
else pair :=false;
…
end.
```

⇒

```
var pair : boolean;
    nb: integer;
begin
…
pair := (nb mod 2 =0);
…
end.
```

Rappel

- Boucles :

En pseudo : tantque (condition) faire instructions; fin tanque

En pascal : while (condition) do instructions;

condition est une expression booléenne

Exemple :

```
var a,b, reste: integer;
begin
  a:=36;
  b:=24;          } Conditions initiales
  while (b<>0) do
  begin           <- Condition d'arrêt
    reste:=a mod b;
    a:= b;        } Corps de la boucle
    b:=reste;
  end;
 writeln(' Le pgcd(',a,' ,',b ')=',a);
```

Procédures et fonctions

- **Quelques citations** :
  - « *Pour comprendre un programme d 'un seul tenant, un être humain met normalement un temps qui augmente exponentiellement avec la taille du programme* » **Djikstra**
  - «*Pour maîtriser la complexité du problème à résoudre, diviser chacune des difficultés que j'examinerais en autant de parcelles qu'il pourrait et qu 'il serait requis pour mieux le résoudre* » **Descartes**
    - **stratégie "diviser pour régner »**

Procédures et fonctions

- **Si vous avez un gros programme à mettre au point, et que certaines parties sont semblables, ou d'autres très complexes , alors il faut le structurer et chercher à utiliser au maximum "l'existant".**
  - **utiliser une procédure (sous-programme) si un même traitement est effectué à plusieurs reprises dans le programme.**

    ↳ permet d'alléger le programme, de faciliter sa maintenance, de le structurer et d 'améliorer sa lisibilité.

Les procédures et les fonctions sont à la base de la programmation structurés

Procédures et fonctions

- **Besoin de procédure et fonctions**

  L'analyse d'un problème (démarche descendante) consiste à découper un problème complexes en **tâches** (sous-problème) que l'on est capable de définir.
  - **Notion de tache** : **une tache est une action bien définie plus ou moins complexe qui s 'exerce à un instant donnée sur un ou plusieurs objets.**
  - ***Exemple*** :
    - ***Échanger*** **le contenu des variables entières A et B.**
    - ***Diviser*** **l 'entier A par l'entier B pour obtenir le quotient Q et le reste R**
    - ***Élever*** **A à la puissance B et mettre le résultat dans C**
    - ***Trier*** **la suite de N éléments contenu dans le tableau T**
    - …

Procédures et fonctions

- **Quand réaliser une procédure**

  On doit réaliser une procédure à chaque fois que l 'analyse d 'un problème conduit à identifier une tâche.
  - **Spécifier une procédure** : **c 'est identifier** :

    1) l'action réalisé par cette procédure

    2) les paramètres formels, c 'est à dire :

    a) l 'ensemble des données nécessaires, ainsi que leur type

    b) l 'ensemble des résultats produits, ainsi que leurs types

Procédures et fonctions

- **Quand réaliser une fonction?**

  En général, le rôle d'une fonction est le même que celui d'une procédure.

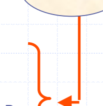
  Il y a cependant quelques différences :
  - renvoi d'une valeur unique
  - la fonction est typé
    - Type de la fonction : scalaire, réel, intervalle, string, structure
- **Exemple 1 :**

  F : R -> R

  x --> ax + b

  Incorporation dans une expression :

  if F(x0) > 10 then …
- **Retour à l'exemple de tâches** :
  - ***Échanger*** le contenu des variables entières A et B.
  - ***Diviser*** l 'entier A par l'entier B pour obtenir le quotient Q et le reste R
  - ***Trier*** la suite de N éléments contenu dans le tableau T
  - ***Élever*** A à la puissance B





Procédures et fonctions

- **Déclaration :**

En pseudo :

procédure nomProc(don_1:type,…,don_n:type, r rés_1,…,rés_m:type)
début
…
fin

fonction nomFonct(don_1:type,…,don_n:type) : typeValeurRetournée
début
…
fin

En pascal :

procedure nomProc(don_1:type,…,don_n:type, var rés_1,…,rés_m:type)
begin
…
end;

function nomFonct(don_1:type,…,don_n:type) : typeValeurRetournée
begin
…
nomFonct:= …
end;

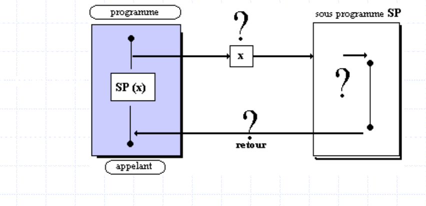
Procédures et fonctions : exemples

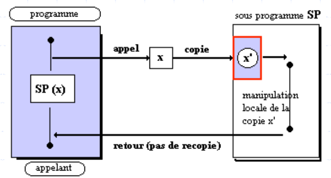
```
procedure echanger(var a,b:integer);
{ données: a,b:entier
 résultats: a,b:entier
 spécifications : échanger le contenu des variables entières A et B.}
var aux:integer;
begin
    aux:=a; a:=b; b:=aux;
end;
procedure division(a,b:integer; var q,r :integer);
{données: a,b:entier
 résultats: q,r:entier
spécifications : Diviser l 'entier A par l'entier B pour obtenir le quotient Q et le reste R
begin
    r:=a; q:=0;
    while (r>=b) do begin
      r:=r-b;   q:=q+1;
    end;
end;
```

Procédures et fonctions : exemples

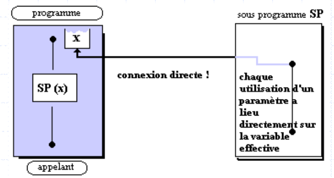
```
function puissance(a,b:integer) : integer;
{ données: a,b:entier
 résultat: a puissance b
 spécifications : élever A à la puissance B.}
var p,i :integer;
begin
    p:=1;
    for i:=1 to b do
      p:=p*a;
puissance := p;
end;
```

Procédures et fonctions : passage de paramètres






Passage par valeur



Passage par adresse (par référence)

Procédures et fonctions : utilisation

- **Utilisation d 'une procédure et passage de paramètres**

```
program appels_procs;
var x,y, quotient, reste :integer;
procedure division(a,b:integer; var q,r :integer);
{ données: a,b:entier
  résultats: q,r:entier
  spécifications : Diviser l 'entier A par l'entier B pour obtenir le quotient Q et le reste R}
begin
    r:=a;   q:=0;
    while (r>=b) do begin
      r:=r-b;   q:=q+1;
    end;
end;

begin
writeln(' division entière de x par y ');
write(' x? ');readln(x);
write(' y? ');readln(y);
division(x,y,quotient,reste);
writeln('le quotient de la division de ',x, 'par ',y, 'est ' , quotient, ' le reste est ', reste );
end.
```

x	y	quotient	reste
16	6	2	4

a	b	q	r
16	6		

Procédures et fonctions : utilisation

- **Utilisation d 'une fonction**

```
program appels_foncts;
var x,y:integer;
function puissance(a,b:integer) : integer;
{ données: a,b:entier
  résultat: a puissance b
  spécifications : élever A à la puissance B.}
var p,i :integer;
begin
   p:=1;
   for i:=1 to b do
      p:=p*a;
   puissance := p;
end;
begin
   writeln(' calcul de x puissance y ');
   write(' x? ');readln(x);
   write(' y? ');readln(y);
   writeln('puissance (',x, ', ',y, ')= ', puissance(x,y) );
end.
```

On peut écrire :

z:=2 * puissance(x,y)-10;

Procédures et fonctions : exemple

Programme Pascal

```
Program convertir_repeat (input, output) ;
    const  POINT = '.' ;
               BASE = 10 ;
    var  resultat : real ;
          partEntiere,partDecimale,nbChiffre : integer ;
          car : char ;
function estUnChiffre( car:char):boolean;
begin
    estUnChiffre:=('0'<=car) and (car<='9') ;
end;
function car2Chiffre( car:char):integer;
begin
    car2Chiffre:=ord(car)-ord('0') ;
end;
procedure repererUnNombre(var car:char)
begin
    repeat
            read(car)
    until (estUnChiffre(car)) ;
end;
```

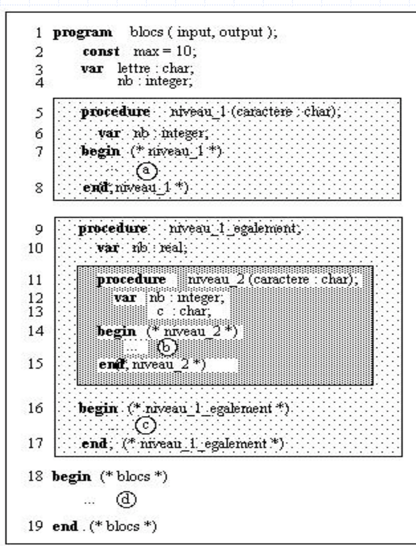
Procédures et fonctions : exemple

```
procedure extraireNombre(var car : char; var nb, nbChiffre:integer);
begin
    nb := 0;nbChiffre:=0;
    repeat
           nbChiffre:=nbChiffre+1;
           nb := BASE*nb +  car2chiffre(car);
           read(car)
    until (not (estUnChiffre(car));
end;
fonction miseAEchelle(partDecimale,nbChiffre:integer): real;
var nb:real;
begin
    nb:=partDecimale;
    while (nbChiffre >0) do
    begin
           nb := nb/BASE ;
           nbChiffre := nbChiffre -1
    end
    miseAEchelle:=nb;
end;
```

Procédures et fonctions : exemple

```
begin
        repererUnNombre(car);
        extraireNombre(car, partEntiere,nbChiffre);
        if (car = POINT) then
        begin
                read(car) ;
                extraireNombre(car, partDecimale,nbChiffre);
                resultat:=partEntiere+miseAEchelle(partDecimale,nbChiffre):
        end ;
        writeln(resultat)
end.
```

Visibilités variables locales et globales



```
program   blocs ( input, output );
    const   max = 10;
    var   lettre : char;
          nb : integer;

    procedure   niveau_1 (caractere : char);
        var   nb : integer;
    begin   (* niveau_1 *)
        ... ⓐ
    end; (* niveau_1 *)

    procedure   niveau_1_egalement;
        var   nb : real;

        procedure   niveau_2 (caractere : char);
            var   nb : integer;
                  c : char;
        begin   (* niveau_2 *)
            ... ⓑ
        end; (* niveau_2 *)

    begin   (* niveau_1_egalement *)
        ... ⓒ
    end;   (* niveau_1_egalement *)

begin   (* blocs *)
    ... ⓓ
end . (* blocs *)
```

Règles de visibilité - Objets locaux ou globaux

```
program tri_suite;
const MAX = 100;
type suiteEntier = record
        nbElement : integer;
        elements : array[1..MAX] of integer;
     end;

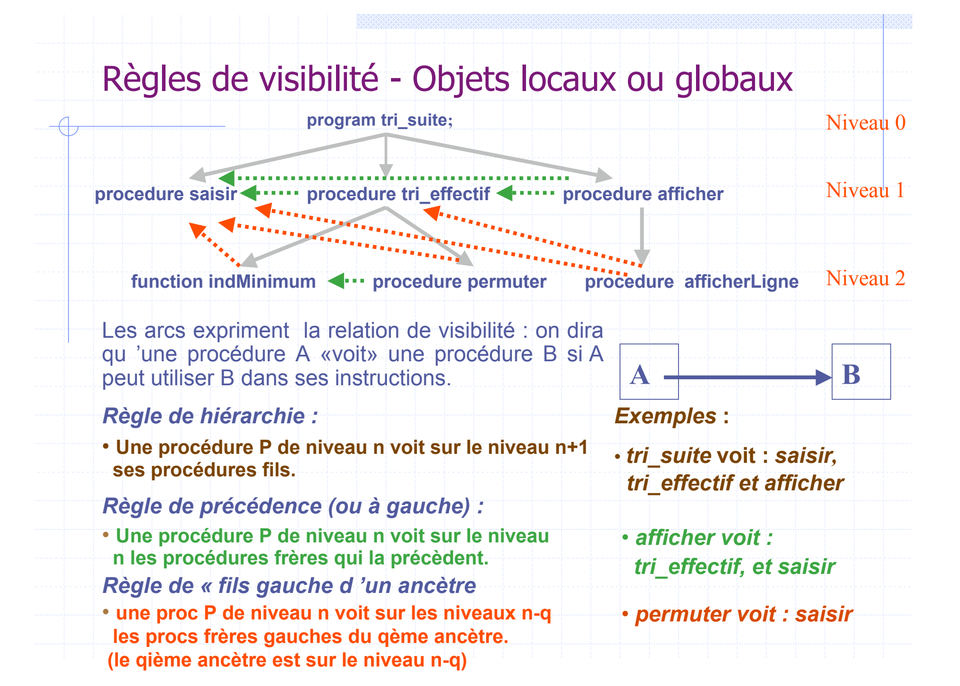
  procedure saisir(var s:suiteEntier)
  ...

  procedure tri_effectif(var s:suiteEntier);
  ...
     function indMinimum(s:suiteEntier, pos:integer):integer;
     ...

     procedure permuter(var a, b : integer);
     ...
  begin
  ...
  end;

  procedure afficher(s:suiteEntier)
  ...
     procedure  afficherLigne(s:suiteEntier, lg:integer);
     ...
  begin
  ...
  end;
begin
...
end.
```

Règles de visibilité - Objets locaux ou globaux



Les arcs expriment la relation de visibilité : on dira qu 'une procédure A «voit» une procédure B si A peut utiliser B dans ses instructions.

Règle de hiérarchie :

- **Une procédure P de niveau n voit sur le niveau n+1 ses procédures fils.**

Règle de précédence (ou à gauche) :

- **Une procédure P de niveau n voit sur le niveau n les procédures frères qui la précèdent.**

Règle de « fils gauche d 'un ancètre

- **une proc P de niveau n voit sur les niveaux n-q les procs frères gauches du qème ancètre. (le qième ancètre est sur le niveau n-q)**

Exemples :

- ***tri_suite voit : saisir, tri_effectif et afficher***
- ***afficher voit : tri_effectif, et saisir***
- ***permuter voit : saisir***

Visibilité, localité et globalité des objets

- **Définitions** :
  - on appelle objet, une constante, une variable ou un paramètre formel.
  - On appelle objet local à une procédure, un objet défini dans cette procédure
  - on appelle objet global à une procédure P, un objet défini dans les procédures ancêtres de P (sur le chemin menant de P à la racine).
  - on appelle objet global à un programme, un objet défini au niveau 0.

Visibilité, localité et globalité des objets

- **Règle de visibilité des objets** :
  Dans une proc P de niveau n, les objets utilisables sont les objets globaux et les objets locaux à P.
  - Dans le cas où un objet local à une proc P, et un objet global à une proc P portent le même nom, l 'objet qui sera considéré est l 'objet local.
  - Plus généralement, dans le cas où deux objet globaux à la proc P portent le même nom, l 'objet qui sera considéré est l 'objet le plus proche de P (celui défini dans l 'ancêtre de P le plus proche).
- **Remarque :**
  les règles et définitions précédentes sont également valables dans le cas des fonctions.

Visibilité, localité et globalité des objets

- **Pricipe de la « boîte noire »**
  Une procédure peut être assimilé à une « boîte noire », c'est à dire un module dont on peut ne connaître que les interactions possibles avec le monde extérieur :



Les règles de visibilités au niveau des objets permettent d 'utiliser des objets globaux; il s 'agit alors de la programmation par effet de bords. Ceci à parfois un intérêt, mais :
  - C'est totalement déconseillé au programmeur débutant.
  - À éviter pour le programmeur expérimenté.

Visibilité, localité et globalité des objets

- ***Local ou global ?***
  Il vaut toujours mieux privilégier les variables locales aux variables globales.
  - **Inconvénients d'une utilisation systématique de variables globales**
    - manque de lisibilité
    - risque d'effets de bord si la procédure modifie les variables globales
  - **Avantages d'une utilisation de variables locales**
    - meilleure structuration
    - diminution des erreurs de programmation
    - les variables locales servent de variables intermédiaires (tampon) et sont "oubliées" (effacées de la mémoire) à la sortie de la procédure

Une procédure doit effectuer la tâche qui lui a été confiée,

en ne modifiant que l'état de ses variables locales.

Exemple

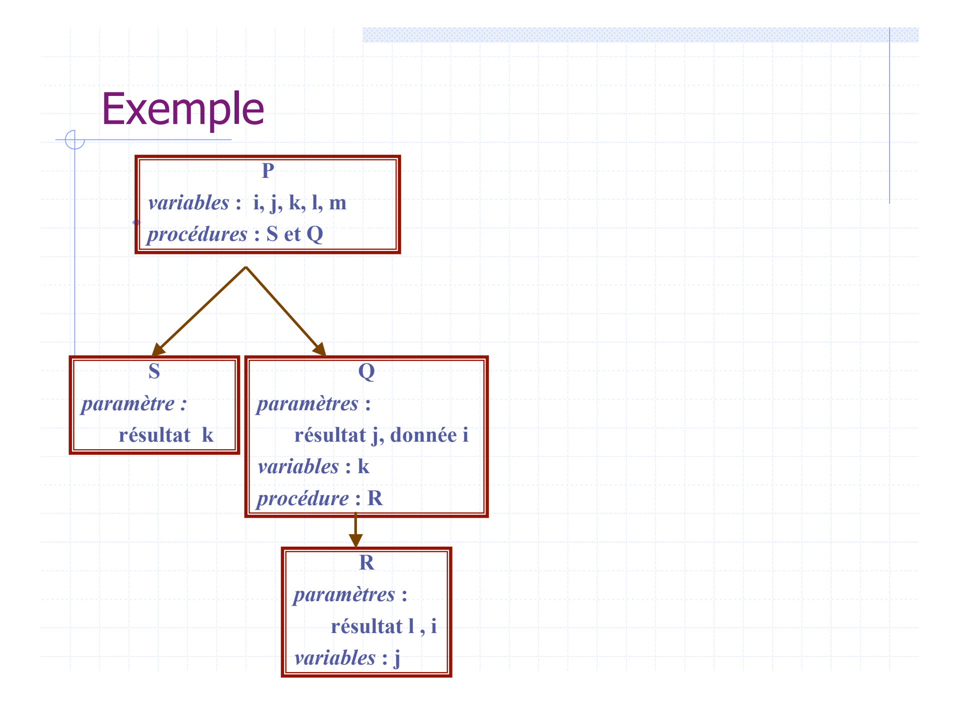
- Dérouler à la main le programme Pascal suivant.
- Indiquez les valeurs affichés à l'écran.
- À chaque occurrence du symbole ✓ dans la partie instruction de ce programme,

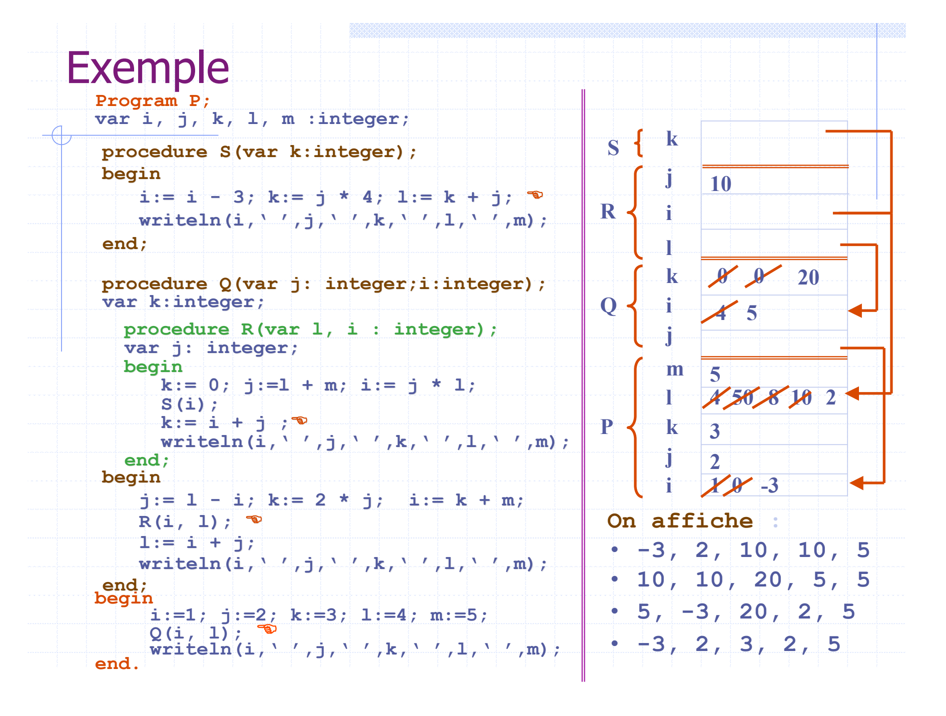
  indiquez quel est l 'état de la pile à l'éxécution.

Exemple

```
Program P;
var i, j, k, l, m :integer;
procedure S(var k:integer);
begin
    i:= i - 3; k:= j * 4; l:= k + j; ✓
    writeln(i,` ´, j, ` ´, k,` ´, l,` ´, m);
end;
procedure Q(var j: integer; i:integer);
var k:integer;
    procedure R(var l, i : integer);
    var j: integer;
    begin
      k:= 0; j:=l + m; i:= j * l;
      S(i);
      k:= i + j ; ✓
      writeln(i,` ´, j, ` ´, k,` ´, l,` ´, m);
    end;
begin
j:= l - i; k:= 2 * j;  i:= k + m;
R(i, l);
l:= i + j;  ✓
writeln(i,` ´, j, ` ´, k,` ´, l,` ´, m);
end;
```

```
begin
i:=1; j:=2; k:=3; l:=4; m:=5;
Q(i, l); ✓
writeln(i,` ´, j, ` ´, k,` ´, l,` ´, m);
end.
```





Récursivité

Raisonnement par récurrence

- Exemple : Montrer que : $S(n) = \sum_{i=0}^{n} i = \frac{n(n+1)}{2}$

  Preuve :

  Cas de base :

  - *n = 0 :* $S(0) = \sum_{i=0}^{0} i = 0 = \frac{0(0+1)}{2} = 0$ *(Vrai)*
  - *n = 1 :* $S(1) = \sum_{i=0}^{1} i = 0 + 1 = \frac{1(1+1)}{2} = 1$ *(Vrai)*

  Récurrence :

  Hypothèse : S(n) est vrai,

  ➢ *montrer que S(n+1) est vrai*

Récursivité

hypothèse de récurrence

⇩

$$S(n+1) = \sum_{i=0}^{n+1} i = (\sum_{i=0}^{n} i) + (n+1) = S(n) + (n+1)$$

$$= \frac{n(n+1)}{2} + (n+1)$$

$$= (n+1)(\frac{n}{2}+1) = \frac{(n+1)(n+2)}{2} \quad (Vrai)$$

Récursivité

Équations de récurrences

- Base : S(0) = 0
- Récurrence : S(n+1) = S(n)+(n+1)

☞ La récursivité est un mécanisme de calcul puissant!!

↳ Résoudre un *problème de taille n* peut se ramener à la résolution *d'un (plusieurs) sous problèmes de taille plus réduite* ... ⇨

Récursivité

- **Action récursive**

  « une action A est exprimée de façon récursive si la décomposition de A fait appel à A. »

- Action calculer S(n) :

  Appliquer les deux règles suivantes:

  1) S(0) = 0

  2) S(n) = S(n-1) + n , n>0

➢ S(3) ?

= S(2) +3 = (S(1)+2) +3

= ((S(0)+1)+2)+3 = ((0+1)+2)+3 = 6

Récursivité

- **Objet récursif**

  « La définition d 'un objet est récursive lorsqu 'elle se formule en utilisant l'objet même qu'elle entend définir »

- Exemple (déf. récursive d'une chaîne)

  Une chaîne de caractère est :

  - soit la chaîne vide
  - soit un caractère suivi d'une chaîne de caractère

Récursivité

- **Exemple (suite)**

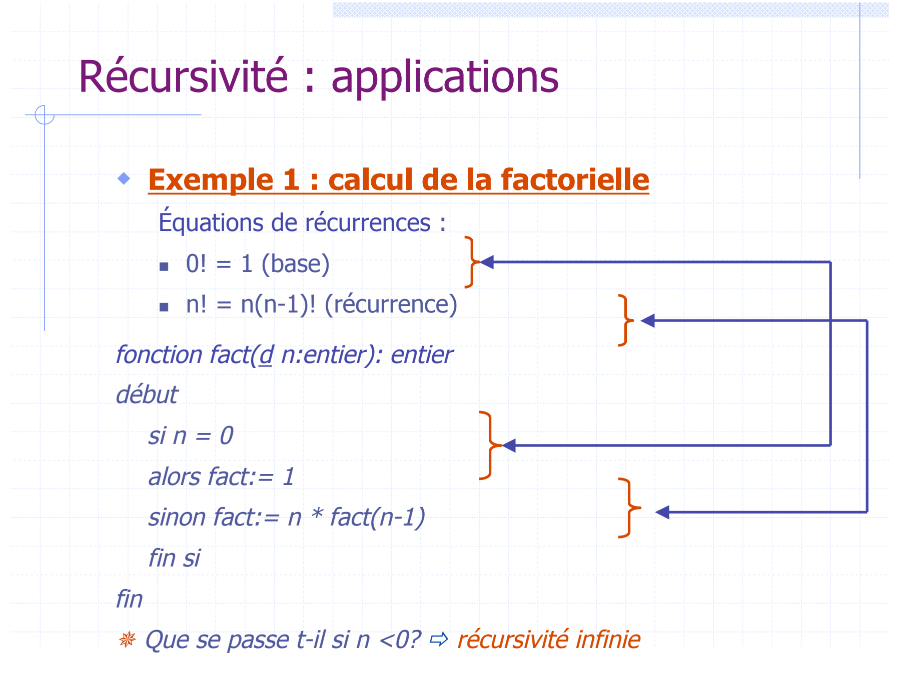
D'après la définition précédente :

- «a» est une chaîne :
  un caractère 'a' suivi d'une chaîne vide
- «ab» est une chaîne :
  un caractère 'a' suivi de la chaîne « b »

☞ La récursivité est un mécanisme puissant de définition d'objets!

Récursivité Vs Itération

Itération	***Récursivité***
procédure Itération( ) *Tantque (condition) faire* *<Instructions>* *fin tantque*	*procédure Itération( )* *Si (condition) alors* *<Instructions>* *Itération()* *fin si*
fonction S(n) : entier *S :=0* *Tant que (n>0) faire* *S:=S+n* *n:=n-1* *fin tant que* *retourner S*	*fonction S(n) : entier* *Si (n=0)* *alors S:= 0* *sinon S:= S(n-1)+n* *fin si*

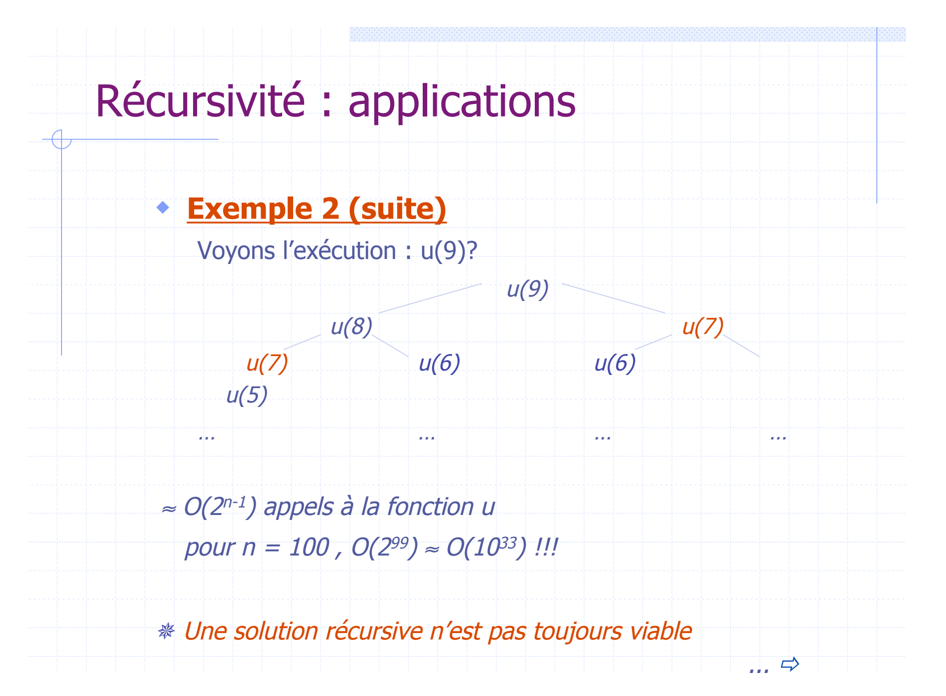


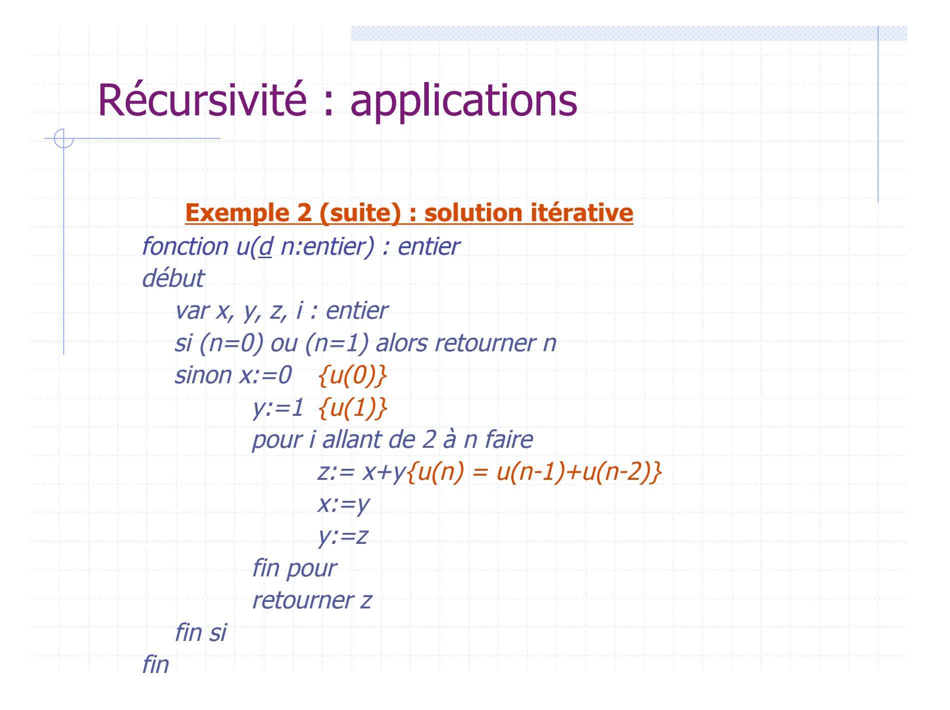
Récursivité : applications

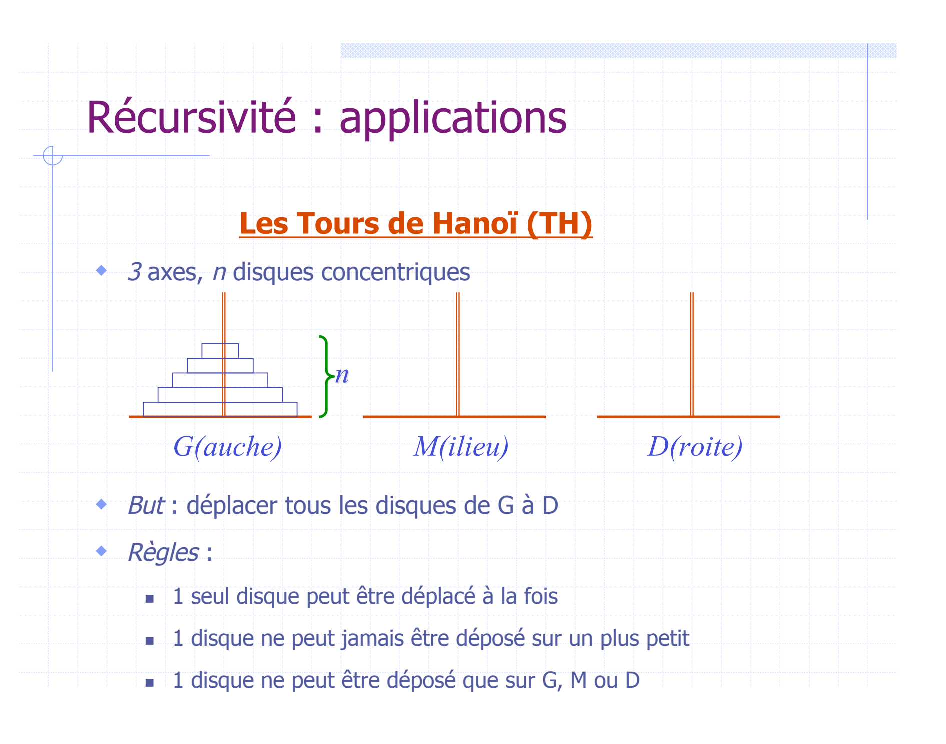
- **Exemple 2 : suite de fibonacci**

  Équations de récurrences :
  - u(0) = 0, u(1) = 1 (Base)
  - u(n) = u(n-1)+u(n-2), n>1 (récurrence)

fonction u(d n:entier) : entier
si (n=0) ou (n=1)
alors u:= n
sinon u:= u(n-1) + u(n-2)
fin si
fin action







Récursivité : applications

TH : idée de la solution récursive

- *exprimer le problème des n disques à l 'aide de la même solution sur n-1 disques*
- *(base) savoir résoudre le problème pour 1 disque*

Équations de récurrences

G(auche) *M(ilieu)* *D(roite)*

(k=1)

Récursivité : applications

1 2 3

G(auche) *M(ilieu)* *D(roite)*

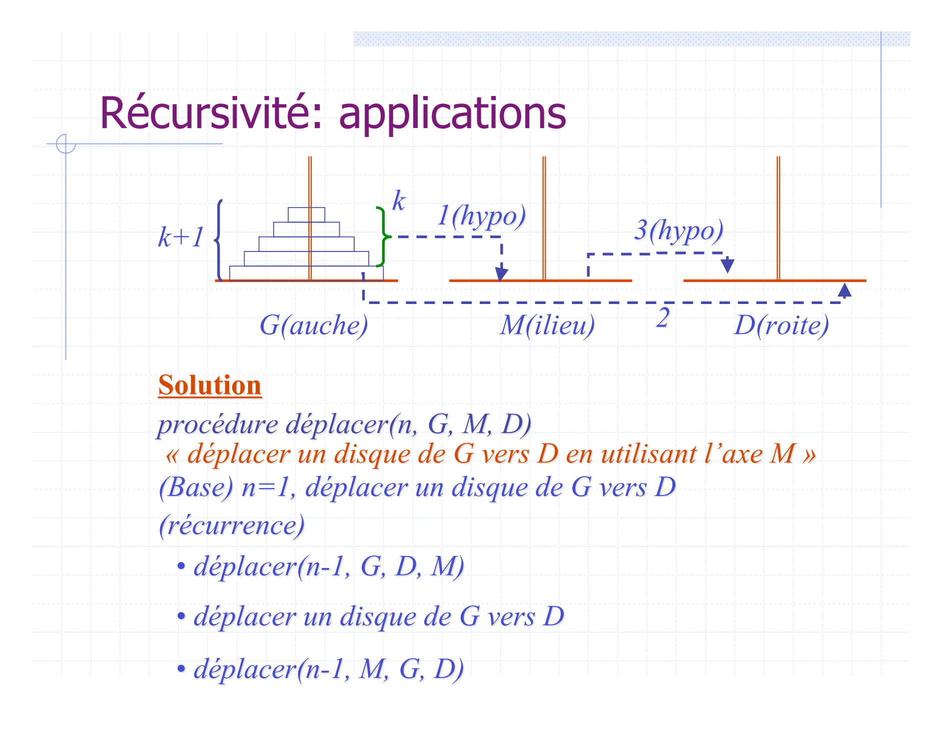
(k=2)

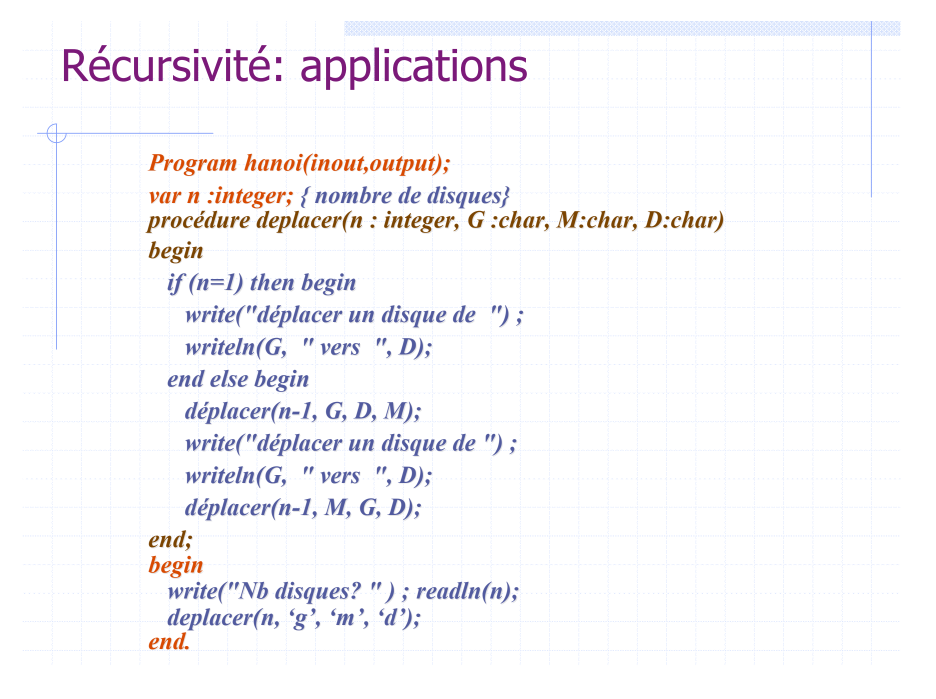
(k=3) …, bonne chance

Récurrence :

Hypothèse: on connaît la solution le problème pour *k* disques

Trouver la solution pour *k+1* disques





Récursivité : pile d'exécution(exemple)

```
Program PP();
var nb = 37, sc : integer;
procedure P(n: entier, var s : entier)
var c : integer;
begin
  if(n = 0)  then s :=0
  else begin
    c := n mod 10;
    P(n div 10, s);
    s:=s +c;
  end;
end;

begin
 P(nb, sc);
 writeln ("somme des chiffres de", nb, "est ", sc)
end.
```

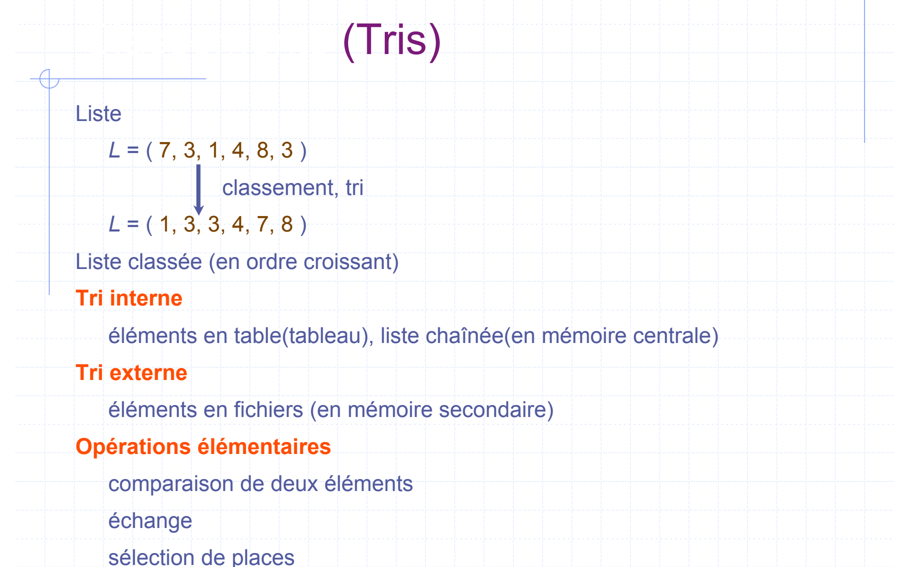
P	c	
	n	0
	s	
P	c	3
	n	3
	s	
P	c	7
	n	37
	s	
PP	nb	37
	sc	? 0 3 10

Pile d'exécution

Récursivité

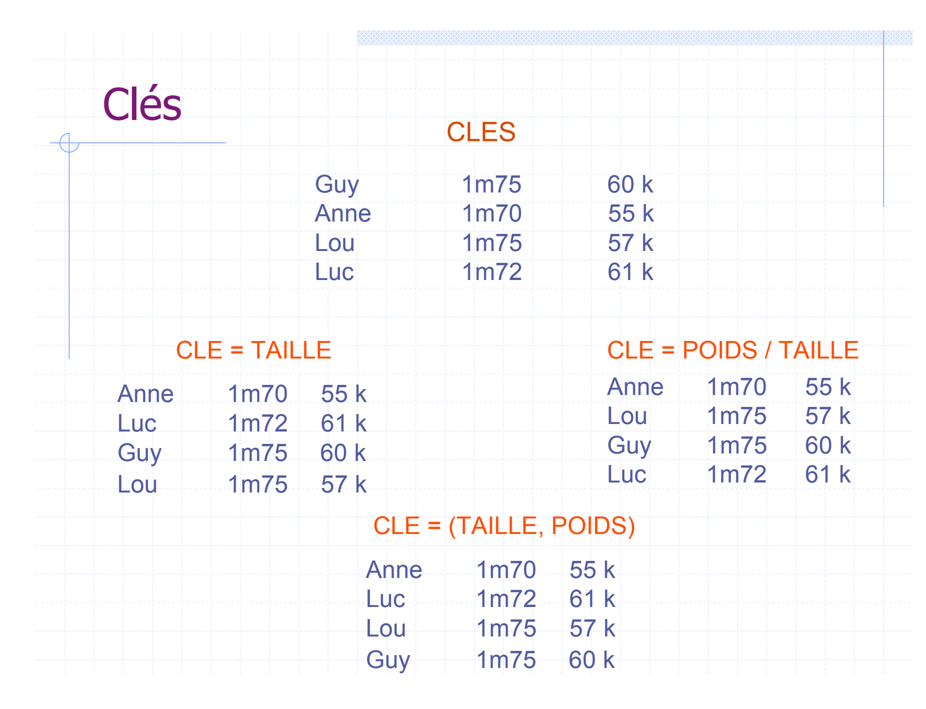
Conclusion

- La récursivité est un principe puissant nous permettant de :
  - définir des structures de données complexes
  - de simplifier les algorithmes opérants sur ces structures de données



Tris internes

- **Elémentaires**
  - Sélection
  - Insertion
  - Bulles
- **Dichotomiques**
  - Tri rapide
  - Fusion



Tri en TABLE

$L = (e_1, e_2, \ldots, e_n)$ en table, accès direct à e_i

Clé : L → Ensemble muni de l'ordre ≤

Problème

Calculer *p*, permutation de $\{1, 2, \ldots, n\}$

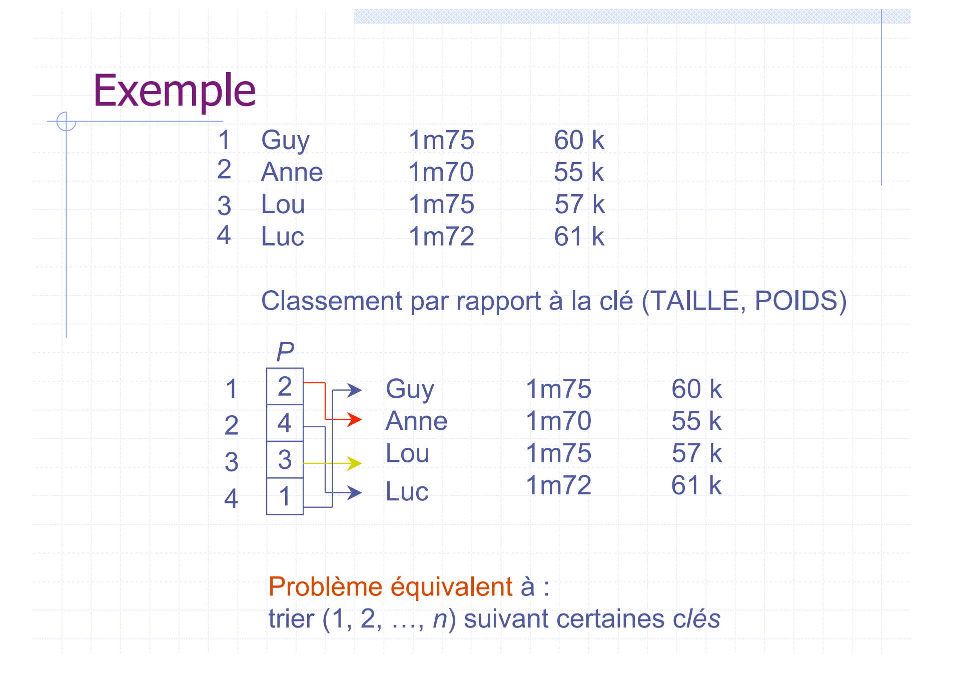
telle que $Clé(\mathbf{e}_{p(1)}) \leq Clé(\mathbf{e}_{p(2)}) \leq \ldots \leq Clé(\mathbf{e}_{p(n)})$

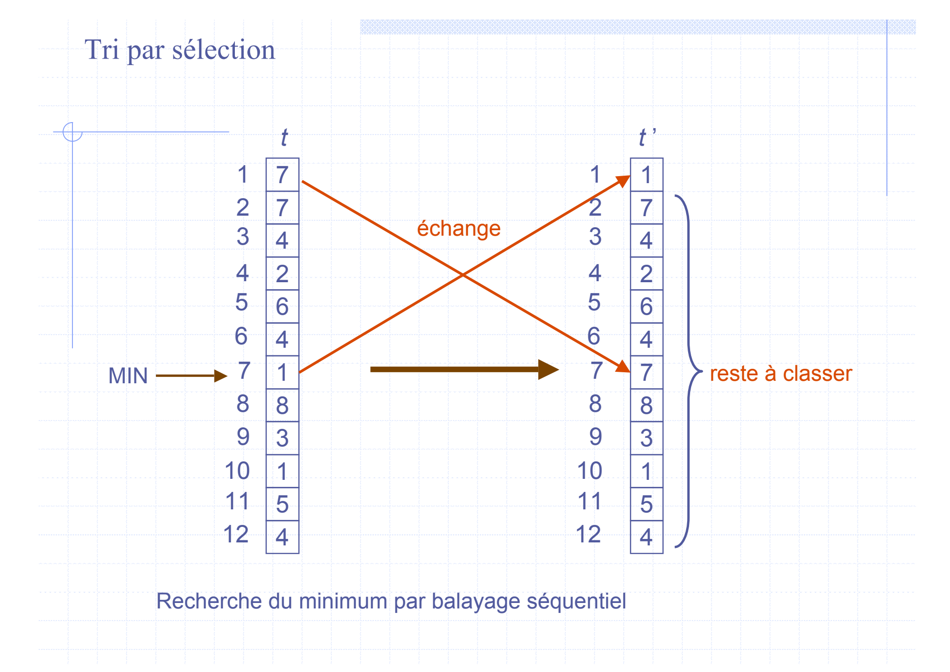
Stabilité

p est stable, chaque fois que $Clé(\mathbf{e}_{p(i)}) = Clé(\mathbf{e}_{p(k)})$:

$i < k$ équivalent $p(i) < p(k)$

[le tri n'échange pas les éléments de même clé]



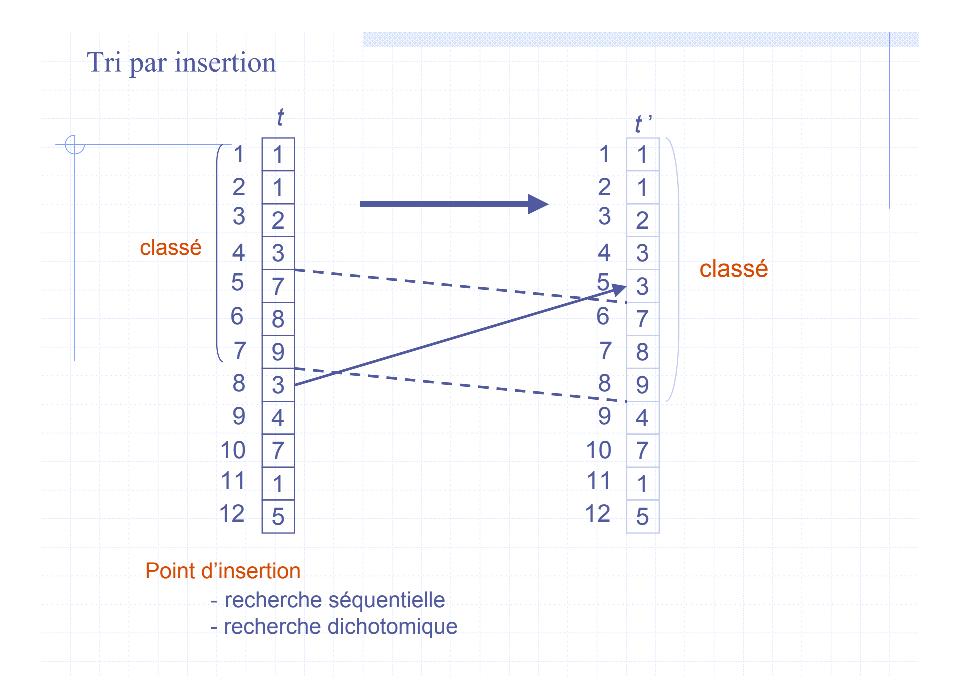


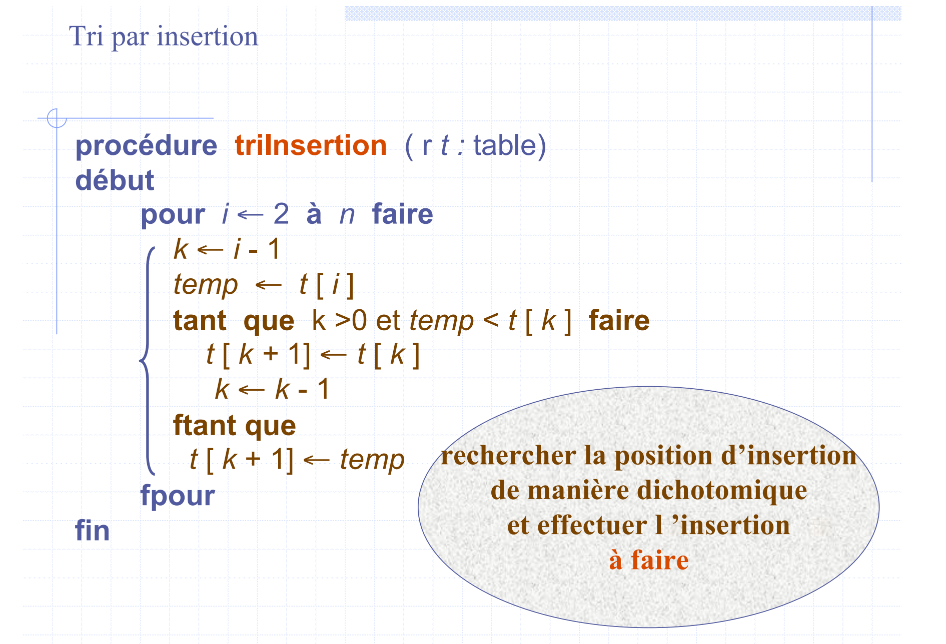
Tri par sélection

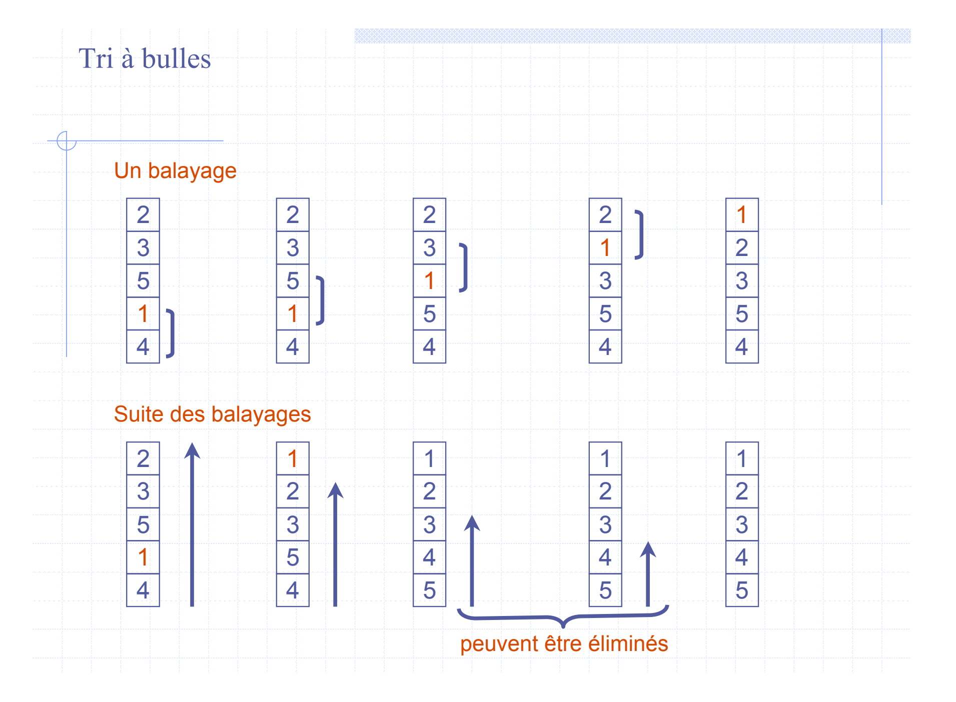
Constante n = ...

Type table = tableau[1..n] de type-element

```
procédure triSelection ( r t :table) ;
var temp : type-element
    min, i, j :entier
début
        pour i variant de 1 à n-1 faire
           min ← i
           pour j ← i + 1 à n faire        }
                si t [ j ] < t [ min ]      }
                alors min ← j               }   ← indMin(t, i,n)
                fsi                         }
           fpour                            }
           echanger (t[ i ], t[min])
        fpour
fin
```





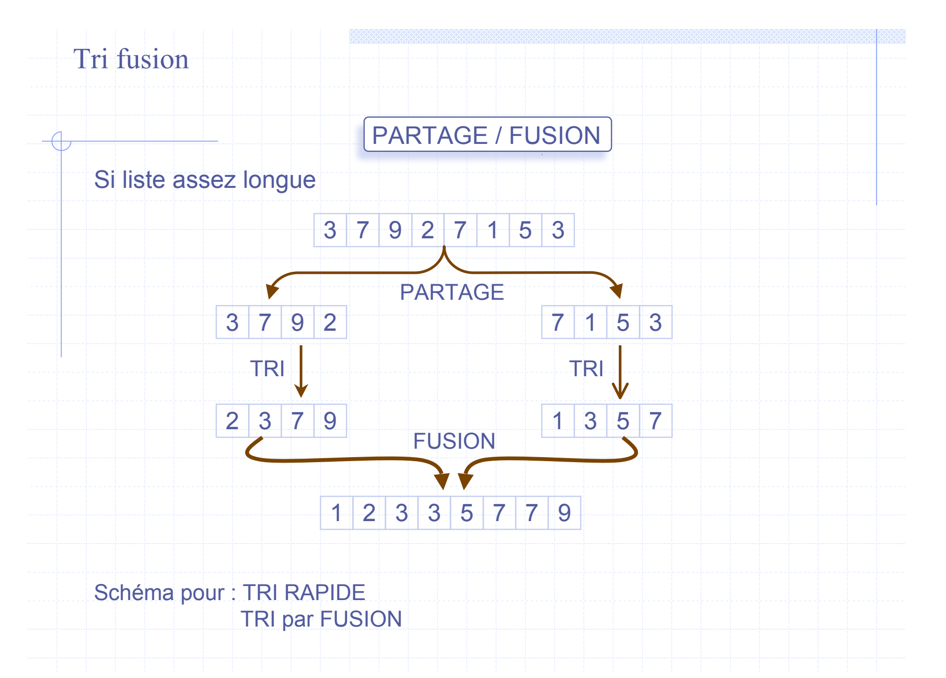


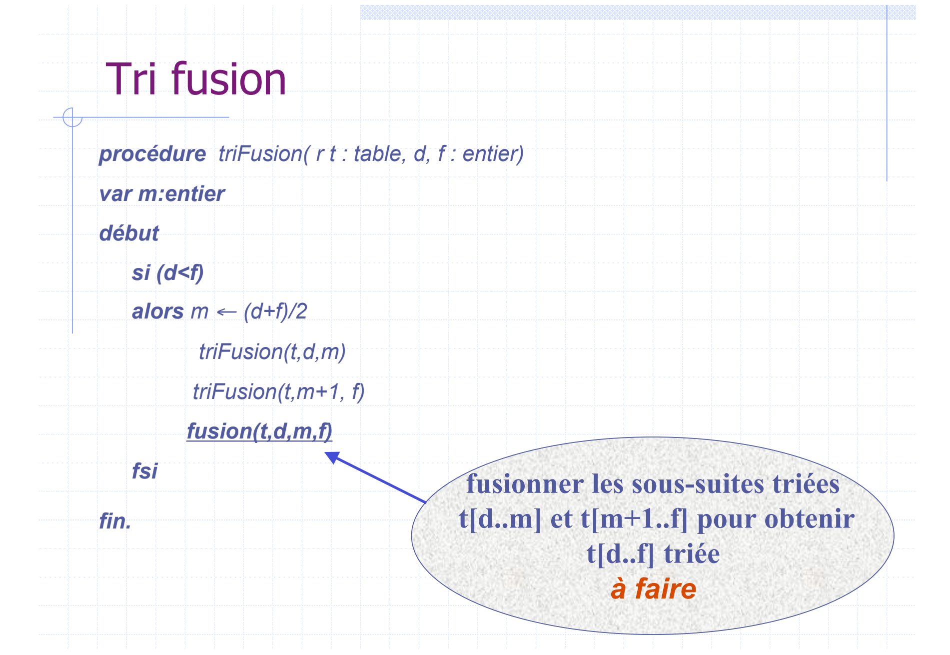
Tris à bulles

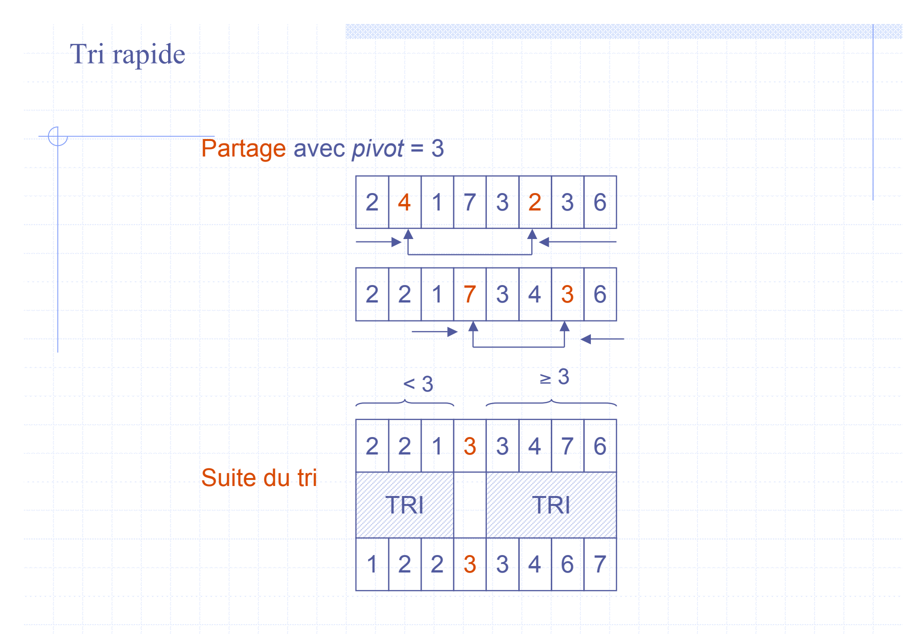
```
procédure triBulles( r t : table)
var i:entier
    changement : booléen
début
    répéter
      changement ← faux
      pour i variant de 1 à n-1 faire
        si t [i] > t [ i+1] alors
                  echanger( t [i], t [i+1] )
                  changement ← vrai
        fsi
    jusqu'à non (changement)
fin
```

Tri à bulles

```
procédure triBulles( r t : table)
var i, k, dernierEchange:entier
début
      i ← 1
      tant que i ≤ n - 1 faire
        dernierEchange ← n
        pour k variant de n à i + 1 pas -1 faire
          si t [ k - 1] > t [ k ] alors
            echanger( t [k-1], t [k] )
            dernierEchange ← k
          fsi
        i ← dernierEchange
      ftant que
fin.
```







Tri rapide

procédure triRapide(r t : table, d, f : entier)
var m:entier
début
- ***si (d<f)***
- ***alors*** *m ← partitionner(t, d, f)*
  - *triRapide(t,d,m-1)*
  - *triRapide(t,m+1, f)*
- ***fsi***

fin.

> **choisir un pivot et**
> **partionner t[d..f]**
> **en deux t[d..m] et t[m+1..f] tq.**
> **t[d..m-1] <=t[m] <=t[m+1..f]**
> ***à faire en TD***

Correction du DS (2002-2003)

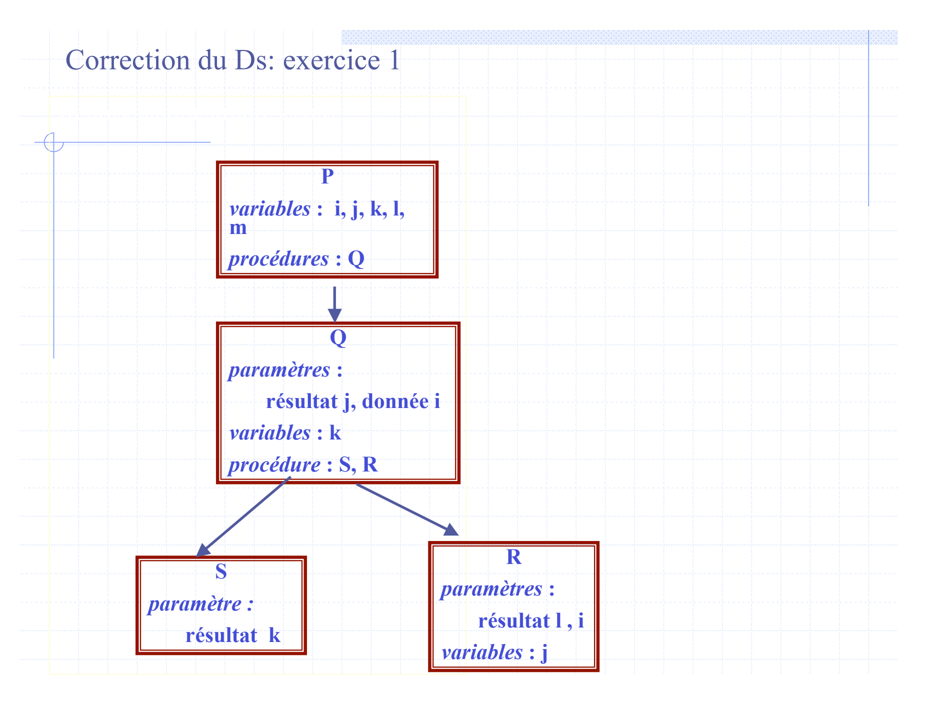
Exercice 1 :
Dérouler à la main le programme Pascal suivant.
Indiquez les valeurs affichés à l'écran.
À chaque occurrence du symbole ✓ dans la partie instruction de ce programme, indiquez quel est l 'état de la pile à l'exécution.

Correction du DS : exercice 1

```
Program P;
var i, j, k, l, m :integer;
procedure Q(var j: integer; i:integer);
var k:integer;
        procedure S(var k:integer);
        begin
              i:= i - 3; k:= j * 4; l:= k + j;  ✓
              writeln(i, ‘ ’, j, ‘ ’, k, ‘ ’, l, ‘ ’, m);
        end;
        procedure R(var l, i : integer);
        var j: integer;
        begin
              k:= 0; j:=l + m; i:= j * l;
              S(i);
              k:= i + j ; ✓
              writeln(i, ‘ ’, j, ‘ ’, k, ‘ ’, l, ‘ ’, m);
        end;
begin
j:= l - i; k:= 2 * j;  i:= k + m;
R(i, l);
l:= i + j; ✓
writeln(i, ‘ ’, j, ‘ ’, k, ‘ ’, l, ‘ ’, m);
end;
```

```
begin
i:=1; j:=2; k:=3; l:=4; m:=5;
Q(i, l); ✓
writeln(i, ‘ ’, j, ‘ ’, k, ‘ ’, l, ‘ ’, m);
end.
```

Correction du Ds: exercice 1



Correction du Ds: exercice 1

```
Program P;
var i, j, k, l, m :integer;
  procedure Q(var j: integer;i:integer);
  var k:integer;
   procedure S(var k:integer);
   begin
      i:= i - 3; k:= j * 4; l:= k + j;
      writeln(i,' ',j,' ',k,' ',l,' ',m);
   end;
  procedure R(var l, i : integer);
  var j: integer;
  begin
     k:= 0; j:=l + m; i:= j * l;
     S(i);
     k:= i + j ;
     writeln(i,' ',j,' ',k,' ',l,' ',m);
  end;
  begin
     j:= l - i; k:= 2 * j;  i:= k + m;
     R(i, l);
     l:= i + j;
     writeln(i,' ',j,' ',k,' ',l,' ',m);
  end;
begin
     i:=1; j:=2; k:=3; l:=4; m:=5;
     Q(i, l);
     writeln(i,' ',j,' ',k,' ',l,' ',m);
end.
```

S	k	
R	j	10
	i	
	l	
Q	k	~~0~~ ~~0~~ ~~0~~ 10
	i	~~4~~ ~~5~~ 2
	j	
P	m	5
	l	~~4~~ ~~50~~ ~~0~~ ~~0~~ 2
	k	3
	j	2
	i	~~1~~ 0

On affiche :
- 2, 0, 0, 0, 5
- 0, 10, 10, 2, 5
- 2, 0, 10, 2, 5
- 0, 2, 3, 2, 5

Correction du Ds: exercice 2

- **Exercice 2**

```
const MAX = 10;
type suiteEntiers = record
                        nbElements:entier;
                        tab: array[1..MAX] of integer;
                   end;
```

Écrire les procédure / fonction suivantes :

- qui , étant donnée un entier *X* et une suite d'entiers ordonnée par ordre croissant *S1* calcule la suite *S2* obtenue à partir de S1 en insérant l'élément *X* à la bonne place.

  *procédure **insérer**(X :entier , S1 : suiteEntiers , **r** S2 : suiteEntiers)*

Correction du Ds: exercice 2

```
procedure inserer (x:integer;S1:tabEntiers;var S2:tabEntiers);
  var i,j:integer;
  begin
    i:=1;
    while (i<=S1.nbElement and x>S1.tab[i]) do
    begin
         S2.tab[i]:=S1.tab[i];
         i:=i+1;
    end;
    S2.tab[i]:=x;
    for j:=  i to S1.nbEelement do
      S2.tab[j+1]:=S1.tab[j];
    S2.nbElement:=S1.nbElement+1;
  end;
```

Correction du Ds: exercice 2

- qui, étant donnée une suite *S1* quelconque, calcule la suite ordonnée *S2* des éléments de *S1*.

```
procedure ordonner(S1:tabEntiers; var S2:tabEntiers);
  var i:integer;
  begin
    S2.nbElement:=0;
    for i:=1 to S1.nbElement do
     inserer(S1.tab[i],S2,S2);
  end;
```

- qui, étant donnée deux suite S1 et S2 teste retourne vrai si S1 et S2 contiennent exactement les mêmes éléments et dans le même ordre

  fonction identiques(S1,S2 : suiteEntiers) : booléen

Correction du Ds: exercice 2

```
function identiques(S1,S2:tabEntiers):boolean;
  var i:integer;
      ok:boolean;
  begin
      i:=1;ok:=(S1.nbElement=S2.nbElement);
      while( i<=S1.nbElement and ok) do
      begin
          ok:=(S1.tab[i]=S2.tab[i]);
          i:=i+1;
      end;
      identiques:=ok;
  end;
```

Correction du Ds: exercice 2

qui, étant donnée une suite S, retourne vrai si *S* est ordonnée, faux sinon.

- en utilisant les questions précédentes.
- sans utiliser les questions précédentes.

  fonction est_ordonnée(S : suiteEntiers) : booléen

- avec utilisation des proc/fonct prec}

```
function estOrdonnee(S1:tabEntiers):boolean;
 var S2:tabEntiers;
begin
  ordonner(S1,S2);
  estOrdonne:=identiques(S1,S2);
end;
```

Correction du Ds: exercice 2

- sans utilisation des proc/fonct prec

```
function estOrdonnee(S1:tabEntiers):boolean;
var ok:boolean;
    i:integer;
begin
  ok:=true;i:=1;
  while(i<S1.nbElement and ok) do
  begin
    ok:=(S1.tab[i]<=S1.tab[i+1]);
    i:=i+1;
  end;
  estOrdonnee:=ok;
end;
```

Correction du Ds: exercice 3

Exercice 3 :

- on se propose d'écrire un programme permettant à l'utilisateur (enfant) de réviser ses tables de multiplications.
- Pour ce faire, le programme doit générer aléatoirement des opérations, et les proposer à l'enfant. A chaque réponse de l'enfant, le programme le félicitera (bonne réponse) , ou lui demandera de réessayer (mauvaise réponse).
- Le programme s'arrêtera après un certain nombre d'opérations réussies. Ce nombre est fixé à l'avance par le programme.

Q1. **Écrire le programme Pascal correspondant.**

Q2. **Que faut-il modifier pour éviter les opérations : a*1 et 1*a**

Correction du Ds: exercice 3 (Q1 et Q2)

```
program tablesMult(iput,output);
uses crt;
const NB_MULT = 50;
var a,b, repEnfant:integer; fin,ok:boolean;
     nbEssaisOk:integer;
begin
  randomize;nbEssaisOk:=0;
  repeat
   a:= random(9)+1;  } Q1  <--  a:=random(8)+2;   Q2
   b:= random(9)+1;  }          b:=random(8)+2;
   ok:=false;
   repeat
         clrscr;
         write(a, ' x ', b, ' = ? ');
         readln(repEnfant);
         ok:=(repEnfant=a*b);
         if(ok) then writeln('Bravo!!')
         else writeln('C''est Faux, r,essayer!!');
         readkey;
   until ok;
   nbEssaisOk:=nbEssaisOk+1;
  until nbEssaisOk=NB_MULT;
end.
```

Correction du Ds: exercice 3 (Q3)

Q3 : **On désire ensuite disposer d'un moyen de détecter l'opération qui lui a posé le plus de problèmes (nombre le plus élevé d'essais successifs avant de trouver la bonne réponse).**

- **Modifier le programme pour afficher une telle opération.**

Correction du Ds: exercice 3 (Q3)

```
program tablesMult(iput,output);
uses crt;
const NB_MULT = 50;
var
 a,b,x,y, nbErreurs, :integer;
 nbErreursMax, repEnfant:integer;
 fin,ok :boolean;
 nbEssaisOk:integer;
begin
  nbEssaisOk:=0;
  randomize;
  nbErreursMax:=0;
  repeat
   a:=random(8)+2;
   b:=random(8)+2;
   nbErreurs:=0;
   ok:=false;
repeat
         clrscr;
         write(a, ' x ', b, ' = ? ');
         readln(repEnfant);
         ok:=(repEnfant=a*b);
         if(ok) then writeln('Bravo!!')
         else begin
               nbErreurs:=nbErreurs+1;
               writeln('Faux!!');
              end;
         readkey;
until ok;
if(nbErreurs>nbErreursMax)
then begin
       nbErreursMax:=nbErreurs;
       x:=a;y:=b;
     end;
nbEssaisOk:=nbEssaisOk+1;
until nbEssaisOk=NB_MULT;
writeln(x,'x',y,'= ', x*y, ' est l''op
la plus difficile pour vous!!');
writeln('Notez bien le résultat!!');
end.
```

Correction du Ds: exercice 3 (Q4)

Q4 : .

- On veut disposer, maintenant d'un moyen de sauvegarder le nombre d'erreurs effectuées sur chaque opération.
- Proposer une nouvelle version du programme, permettant de sauvegarder dans la structure de donnée le nombre d'erreurs pour chaque opération posés. (pour une opération posé à plusieurs reprises par le programme ; le nombre d'erreurs est le cumul des erreurs)

Correction du Ds: exercice 3 (Q4)

```
program tablesMult(iput,output);
uses crt;
const NB_MULT = 50;
type tableMultiplications = array[1..9,1..9]of integer;

var a,b,x,y, i,j, nbErreurs, repEnfant:integer;
    fin,ok :boolean;
    nbEssaisOk:integer;
    table:tableMultiplications;
```

Correction du Ds: exercice 3 (Q4)

```
begin
  for i:= 1 to 9 do
   for j:= 1 to 9 do
     table[i,j]:=0;
  randomize; nbEssaisOk:=0;
  repeat
   a:=random(8)+2;b:=random(8)+2;
   nbErreurs:=0;ok:=false;
   repeat
     clrscr;
     write(a, ' x ', b, ' = ? ');
     readln(repEnfant);
     ok:=(repEnfant=a*b);
     if(ok)
     then writeln('Bravo!!')
     else begin
            nbErreurs:=nbErreurs+1;
            writeln('C''est Faux, r,essayer!!');
          end;
     readkey;
   until ok;
   table[a,b]:=table[a,b]+nbErreurs;
   nbEssaisOk:=nbEssaisOk+1;
  until nbEssaisOk=NB_MULT;

  writeln('Les erreurs effectuees : ');
  write('    ');
  for i:= 1 to 9 do write(i:4);
  writeln;
  for i:=1 to 9 do
  begin
     write(i:4);
     for j:=1 to 9 do
       write(table[i,j]:4);
     writeln;
  end;
end.
```

Un échantillon d'erreurs

- **Procedure inserer(X:integer; S1:suiteEntiers;S2:suitentiers);**
  - i:=1;
    while(X>S1.tab[i]) do ...
  - Pour i de 1 S1.nbElement faire

```
      si X= S1.tab[i]
      alors S2.nbElement:=S1.nbElement
            S2.tab[i]:=S1.tab[i]
      sinon S2.nbElement:=S2.nbElement+1
            tant que X>S1.tab[i] faire
               S2.tab[i]:=S1.tab[i]
            ftant que
            si X<S1.tab[i] faire
               S2.tab[i]:=X
               S2.tab[i+1]:= S1.tab[i]
            fsi
      fsi
  fpour
```

Un échantillon d'erreurs

```
▪ for i := 1 to S1.nbElement do
     if X< S1.tab[i]
     then begin
              pos:=S1.tab[i];
              S2.tab[i]:=S1.tab[i];
              S2.tab[i+1]:=X;
          end
     else  S2.tab[i]:=S1.tab[i];
▪ i:=1; S2.nbElement:= S1.nbElement+1
  repeat
     S2.tab[i]:=S1.tab[i];i:=i+1;
  until S1.tab[i]>=X;
 S2.tab[i]:=X;
  repeat
     S2.tab[i+1]:=S1.tab[i];i:=i+1;
  until i>S1.nbElement;
```

Un échantillon d'erreurs

```
▪ Var p,i:integer;
  begin
   for i := 1 to p-1 do
      S2.tab[i]:=S1.tab[i];
   ...
▪ i:=1; S2.nbElement:= S1.nbElement+1
  repeat
     S2.tab[i]:=S1.tab[i];i:=i+1;
  until S1.tab[i]>=X;
 S2.tab[i]:=X;
  repeat
     S2.tab[i+1]:=S1.tab[i];i:=i+1;
  until i>S1.nbElement;
```

Un échantillon d'erreurs

```
▪ i:= 1;
  repeat
     if X> S1[i]
     then S2[i]:=S1[i]
     else if X<S1[i]
          then begin
                  S2[i]:=X
                  for j:=i to max do
                        S2[j]:=S1[j];
                 end;
     i:=i+1;
     if X>S1[max] then S2[max+1]:=X;
  until i=Max or X> S1[Max];
▪ ....
```

Un échantillon d'erreurs

- **Procedure ordonner(S1:suiteEntiers;var S2:suitentiers);**

```
▪ variable min, i:entier
  min ← i
  pour i allant de  2 à S1.nbelement faire
     si S1.tab[i] < S1.tab[min]
     alors min ← i
          echanger(S1.tab[i],S1.tab[min])
     fsi
  fpour
  pour i allant de 1 à S1.nbElement faire
     S2.tab[i]←S1.tab[i]
  fpour
```

Un échantillon d'erreurs

- **Procedure ordonner(S1:suiteEntiers;var S2:suitentiers);**
  - variable min, a, z:entier

```
min ← S1.tab[1]
a ← 2
tant que a<> S1.nbElement faire
pour i allant de  a à S1.nbElement faire
    si S1.tab[i] < min
    alors   z ← S1.tab[i]
            S1.tab[i] ← min
            min ← z
    fsi
    a ← a+1
fpour
S2.nbElement←S1.nbElement
fpour
```

Un échantillon d'erreurs

- **Procedure ordonner(S1:suiteEntiers;var S2:suitentiers);**
  - variable i, j:entier

```
begin
j ← 0
pour i allant de 2 à S1.nbElement  faire
    si S1.element[i] < S2.element[i-1]
    alors  début
              j ← j+1
              S2.element[j] ← S1.element[i]
           fin
    sinon début
              j ← j+1
              S2.element[j] ← S1.element[i-1]
           fin
    fsi
fpour
fin
```

Un échantillon d'erreurs

- **function estOrdonnée(S:suiteEntiers):boolean;**
  - var i:integer; difference:boolean;

```
begin
i:= 1;
difference:=false:
reapeat
    if S.tab[i] < S.element[i+1]
    then   difference:=true;
    inc(i);
until ( difference :=false or i = S.nbElement);
estOrdonnee := difference
end;
```

Un échantillon d'erreurs

- **fonction identiques(S1,S2:suiteEntiers):booléen;**
  - var i:=integer; id:booléen;

```
début
si S1.nbElement <> S2.nbElement alors id ←faux
sinon id ← vrai; i ← 1;
        tant que (id et i<= S1.nbElement) faire
          si  S1.tab[i] = S2.tab[i]
          alors id ← vrai;
        ftantque
fsi
identiques ← id
```

  - ...

Un échantillon d'erreurs

- **Program multiplication;**
  - ...

```
Repeat
    X = random(N);
    Y = random(N);
    writeln(x, '* ', y, ' ? '); readln(R);
    R1 := x*y;
    if(R=R1) then begin
                    writeln(' Bbravo ');
                    i:=i+1;
                  end
            else begin
                    writeln(' faux '); writeln(x, ' * ',y, '? ');
                    readln(R);
                 end;
until i> Max
```

  - ...

Les fichiers

- **Motivation**

  Toutes les structures de données que nous avons vues ont un point commun:

  - **Résident dans la mémoire principale** de l'ordinateur. (existence limitée à la durée d'exécution du programme)
  - **Besoin de conserver des données** (pour une utilisation future) après l'exécution du programme (exemple : carnet d 'adresses)
  - **la notion de fichier**.

Les fichiers

- **Qu'est-ce qu'un fichier?**

  Un fichier est une collection de fiches;

  Exemple : un fichier dans une bibliothèque, une administration etc.

Trouver une fiche ?

- parcourir le fichier fiche après fiche
- utiliser une clé d'accès, si le fichier est trié selon cette clé (ordre alphabétique, ordre de cotation des livres...).

Les fichiers

- **En informatique** :

  Un **fichier** est une séquence de données du même type enregistrées sur un support informatique (de manière **permanente**).

- Un fichier est :
  - conservés en *mémoire secondaire*

    (disques et bandes magnétiques, disquettes,...)
  - désigné par un nom et possède des attributs tels que date de création, taille,...

Fichiers : organisation et accès

- ***Définition***

L'organisation d'un fichier est la manière dont sont rangés **physiquement** les enregistrements du fichier.

- Le but est d'arranger les enregistrement de manière à y accéder le plus rapidement possible.
- L'organisation est choisie à la création du fichier.
- Le choix d'une organisation correspond à un compromis entre rapidité d'accès et espace de stockage disponible.

Fichiers : organisation séquentielle

- Elle ne permet qu'un seul accès : le séquentiel.

Toutes les informations sont enregistrées de façon **séquentielle** (linéaire) les unes à la suite des autres.

Pour accéder à une information particulière, il faut nécessairement parcourir le fichier à partir du début, et ce jusqu'au moment où cette information est retrouvée.

Fichiers : organisation relative (accès direct)

Chaque enregistrement possède un **numéro**.

On accède à la fiche recherchée en spécifiant ce numéro d'enregistrement.

L'indication d'un numéro permet donc un **accès direct** à l'information ainsi référencée.

Fichiers : organisation indexée

- **Notion d 'index** :
  - Soit un fichier F dont les enregistrements E possèdent une clé C(e.g. nom). Un index permet d 'associer à chaque clé C le rang R de l 'enregistrement E de clé C dans F.
  - L'index est alors une « table des matières » du fichiers
  - On crée des fichiers supplémentaire d'index
  - On parcourt un index pour rechercher une clef. On obtient ainsi l'adresse exacte de l'information recherchée.

Les types de fichiers

- On distingue deux catégories:
  - ***Les fichiers binaires*** contenant du code binaire représentant chaque élément (enregistrement). Ces fichiers ne doivent être manipulés que par des programmes!
  - ***Les fichiers textes*** (appelés aussi imprimables) contenant des caractères et susceptibles d'être lus, éditées, imprimés...

- Eléments attachés aux fichiers
  - On appelle nom interne d 'un fichier, le nom sous lequel un fichier est identifié. (fichier logique)
  - On appelle nom externe d 'un fichier(fichier physique), le nom sous lequel un fichier est identifié en mémoire secondaire.

    Le nom externe est une chaîne de caractère composée de 3 partie :
    - l ' indentification du support
    - le nom du fichier proprement dit
    - un suffixe qui précise le genre (donnée, texte, etc.)

    C:\TP**donnée**.dat
  - On appelle une fenêtre d 'un fichier, une « zone » de la mémoire principale pouvant contenir un enregistrement du fichier

Fichiers séquentiels

- ***Les fichiers séquentiels en Pascal***

  Définition

  Un **fichier Pascal** est une séquence de données de même type et de longueur indéfinie.

  Points techniques :
  - La fin du fichier est repérée par un marqueur de fin de fichiers : emploi de la fonction booléenne **eof** (end_of_file)

    eof(fichier) = vrai si fin de fichier atteinte

    eof(fichier) = faux sinon
  - Longueur du fichier = nombre d'éléments du fichier

    non définie lors de la déclaration du fichier
  - S'il n'y a pas d'éléments : fichier vide

- **Déclaration**

  Un fichier (binaire) se déclare :

  En pseudo :

  type fichierDeTypeElement = **fichier de** type-élément ;

  En Pascal :

  type fichierDeTypeElement = **file of** type-élément ;

- Tous les types sont autorisés pour les éléments, sauf le type fichier !!!

Les types de fichiers

- Exemples :

```
const LG_MAX_NOM = 30;
      LG_MAX_TITRE = 40;
type
      auteurLivre = string [ LG_MAX_NOM ];
      titreLivre = string [ LG_MAX_TITRE ];
      livreCote = record
                      nomAuteur : auteurLivre;
                      titre : titreLivre;
                      cote : integer;
                  end;
      fichierBibliothèque = file of livreCote;
var FichierBib : fichierBibliothèque;
```

Primitives de manipulations de fichiers séquentiels

- ***Etablir un lien entre le nom interne et le nom externe du fichier***

  Assign(nom interne, nom externe); (en Pascal)

  Assigner(nom interne, nom externe); (en Pseudo)

Exemple : assign(FichierBib, ' C:Livres.bib');

Primitives de manipulations de fichiers séquentiels

- ***Création d'un fichier***

  rewrite (nom interne); (Pascal)

  ouvrirEcriture(nom interne); (Pseudo)

  Cette instruction permet d'ouvrir un fichier 'en écriture', c'est-à-dire de **créer** le fichier, et d'autoriser des opérations d'écriture dans ce dernier.

- Différentes possibilités se présentent lors de l'appel de : Rewrite(F) :
  - Le fichier F n'existe pas : **création** du fichier
  - Si le fichier F existe : **effacement** de toutes les données inscrites dans l'ancien fichier F
  - eof(F) devient vrai
  - Positionnement du *pointeur de fichier* (ou fenêtre) au début du fichier vide F

Primitives de manipulations de fichiers séquentiels

- ***Ecriture dans un fichier***

  Syntaxe

  `write(nom interne,E)` (Pascal)

Ecriture de l'enregistrement E dans le fichier logique F.

Le type de E doit être compatible avec le type déclaré par le fichier.

La commande écrit le contenu de E à l'endroit où pointe le la tête de lecture et la déplace d'une position.

Ecriture dans un fichier

- Exemple

```
type fichierBibliothèque = file of livreCote;
var fichierBib : fichierBibliothèque; {nom interne}
    nomFich:string[MAX_CH]; {pour le nom externe}
    E: livreCote;
begin
    write(‘ entrez le nom du fichier? ’); readln(nomFich);
    assign(fichierBib, nomFich);
    rewrite(fichierBib);
    …
    read (E);
    write(fichierBib, E);
```

- ***Ouverture d'un fichier existant en lecture***

  Syntaxe

  `reset(nom interne)` (Pascal)

  `ouvrirLecture(nom interne)` (Pseudo)

- L'effet de reset(F) :
  - ouverture en lecture et positionnement de la tête de lecture en face du premier enregistrement
  - Lecture du premier enregistrement
  - Si le fichier contient au moins un article, eof(F) devient faux
- ***Lecture d'un fichier***

  Read (F, E) a trois effets :
  - Lecture de l'enregistrement pointé par le fichier
  - Affectation de son contenu dans E
  - Déplacement vers l'enregistrement suivant du fichier

Lecture d 'un fichier : exemple

```
Program  LIRE_ADRESSES ;
  Type
    ADRESSE = record
      NOM, RUE, VILLE : string ;
      NUMERO : integer ;
     end;
    FICHIER = file of ADRESSE ;
  Var CARNET : fichier ;
      CLIENT : adresse ;
  begin
    assign (CARNET, 'carnet_adresses') ;
    reset (CARNET) ;
    while not eof (CARNET) do
       begin
        Read(CARNET,CLIENT);
        write ('NOM :', CLIENT.NOM) ;
        write ('NUMERO, CLIENT. NUMERO) ;
        write ('RUE :', CLIENT. RUE) ;
        write ('VILLE :', CLIENT. VILLE) ;
       end ;
    close (CARNET) ;
  end.
```

Ecriture dans un fichier : exemple

```
Program  ADRESSES ;
    Type
      ADRESSE = record
          NOM, RUE, VILLE : string ;
          NUMERO : integer ;
        end;
       FICHIER = file of ADRESSE ;
    Var CARNET : fichier ;
        CLIENT : adresse ;
        C : char ;
    begin
       assign (CARNET, 'C:\Perso\carnet_adresses') ;
       rewrite (CARNET) ;
       C := 'O' ;
       while C <> 'N' do
           begin
              write ('NOM :') ; readln (CLIENT.NOM) ;
              write ('NUMERO :') ; readln (CLIENT.NUMERO) ;
              write ('RUE :') ; readln (CLIENT.RUE) ;
              write ('VILLE :') ; readln (CLIENT.VILLE) ;
              write(CARNET,CLIENT) ;
              writeln ('autre adresse ?') ; readln (C) ;
           end ;
       close (CARNET) ;
end.
```

Les fichiers de Texte

Dans le cas de fichiers de texte, on utilise les déclarations suivantes, qui sont toutes équivalentes, mais varient selon les compilateurs Pascal :

```
Type Fichier = File of char;
Type Fichier = Text;
```

- Comme pour la saisie de données au clavier et pour l'affichage d'informations à l'écran, il est possible d'utiliser les instructions READLN et WRITELN au lieu de READ et WRITE.
- La fonction EOLN (End Of LiNe) est également disponible.

Exemple 1 : *devinez ce que fait ce programme ?*

```
Program MON_TEXTE;
   Var
      F : Text;
      rep : char;
 Procedure ECRIT;
   Var
        c : char;
   Begin
        Assign(F, 'fichierText' );
        Rewrite(F );
        Writeln( ' Tapez un texte et terminez par $' );
        read( c );
        While (c<>'$') do begin
            write(F,c);
            read( c );
         end;
        Close(F);
   End;
```

Exemple 1 : *devinez ce que fait ce programme ?*

```
Procedure LIT;
     Var c : char;
     Begin
        Reset(F, 'fichierText' );
        read(F,c);
        while (not eof(F) ) do begin
          write(c);
          read (F,c);
        end;
        Close(F);
     End;
  Begin
         Repeat
           Writeln( ' 1) Ecriture' );
           Writeln( ' 2) Lecture' );
           Writeln( ' 0) Fin' );
           Write( ' Votre choix => ' );
           Readln(rep);
           Case rep of
             '1' : ECRIT;
             '2' : LIT;
            End;
         Until rep='0';
   End
```

Les fichiers à Accès Direct

- **Quelques instructions**

  soit var f : file of <typElements>

  - **FILEPOS(f)**
    - indique la position du pointeur courant(tête de lecture) donne un entier
  - **FILESIZE(f)**
    - retourne le nombre d 'enregistrements (composants) du fichier
  - **SEEK(f,i)**
    - positionne le pointeur au composant n°i (0≤ i < filesize(f))
  - **IORESULT**
    - variable contenant 0 si l'action sur un fichier s'est bien déroulée, un entier supérieur à 0 sinon

Exemple

- **Exemple**

```
Program  LIRE_PRODUITS_DIRECT;
     Type PRODUIT = record
                    NOM : string[20] ;
                    PRIX : real ;
                end;
      var   ARTICLE : produit ; FICH : file of PRODUIT ; N : integer ;
begin
     assign(FICH, 'inventaire') ;
     {$I-} reset (FICH) ; {$I+}
     if (IOResult <> 0)
     then writeln('Fichier non trouve!!')
     else begin
             writeln('la taille du fichier est : ', filesize(FICH)
              write('Entrez un numéro d'enregistrement? ');readln (N) ;
             SEEK (FICH, N) ;
              read (FICH, ARTICLE) ;
              writeln ('Nom du produit: ', ARTICLE. NOM, 'Prix:', ARTICLE.PRIX) ;
             close(FICH);
       end;
end.
```

Gestion d'un carnet d'adresses

- **BUT** : proposer une solution de gestion d'un ensemble d'informations concernant des personnes (nom, prénom, adresse, code postal, ville, téléphone, e-mail…)*. On désire pouvoir:
  - **créer**,
  - **consulter**,
  - **supprimer et modifier**.

* **Pour information**: il faut savoir que la constitution et l'exploitation d'un fichier informatique contenant des informations sur des individus doit être soumis à l'accord de la **CNIL** (Commission Nationale Informatique et Liberté). Dans le cas contraire, de graves sanctions pénales sont encourues.

Gestion d'un carnet d'adresses

- Proposer les structure de données permettant de :
  - stocker les informations concernant une personne,
  - de stocker durablement toutes les informations relatives à toutes les personnes.

```
{Déclaration des types}
TYPE
{Enregistrement}
        adresse=record
                        nom, prenom,
                        adresse, cp, ville,
                        tel, mail: string;
                end;
{Fichier d'enregistrements}
        carnetAdresse =file of adresse;
```

Gestion d'un carnet d'adresses

- Écrire un programme principal avec menu permettant les opérations suivantes :
  - créer/recréer le fichier contenant les données ;
  - ajouter un ou plusieurs enregistrements ;
  - obtenir la liste des enregistrements à l'écran ou dans un fichier texte ;
  - afficher les renseignements concernant une personne;
  - modifier les données concernant une personne, sachant sa position dans le fichier ;
  - supprimer une personne du fichier;
  - rechercher les renseignements concernant une personne à partir de son nom et de son prénom ;
  - rechercher toutes les personnes habitant une ville donnée ;
  - trier le fichier par ordre alphabétique des noms et des prénoms des personnes.

Menu

```
Function menu():integer;
var choix:integer;
begin
    writeln('1) Creer/recreer le fichier de donnees');
    writeln('2) Ajouter des enregistrements');
    writeln('3) Liste des enregistrements');
    writeln('4) Affichage d''un enregistrement a partir de sa position');
    writeln('5) Modification d''un enregistrement a partir de sa
    position');
    writeln('6) Suppression d''un enregistrement a partir de sa
    position');
    writeln('7) Recherche a partir du nom et du prenom');
    writeln('8) Recherche des personnes habitant une ville');
    writeln('9) Tri par ordre alphabétique');
    writeln( '10) Recherche rapide sur fichier trie');
    writeln;writeln('0) Fin');writeln;writeln;
    write('Votre choix : ');readln(choix);
    menu:=choix;
end;
```

Programme principal

```
program fichiers(iput,output);
var f:carnetAdresse;
...
begin
    assign(f, 'carnet.fic');
    repeat
        choix := menu;
        case choix of
                1: creation(f);
                2: ajout(f);
                3: liste(f);
                4: acces(f);
                5: modif(f);
                6: suppression(f);
                7: recherche_nom(f);
                8: recherche_ville(f);
                9: tri(f);
                10: dicho(f);
        end;
    until choix=0;
end.
```

Création

```
procedure creation(var f: carnetAdresse);
var choix: integer;
begin
    repeat
        write('Detruire le fichier existant (1) ou annuler (2) ? ');
        readln(choix);
    until (choix=1) or (choix=2);
    if choix=1 then
    begin
        rewrite(f);
        close(f);
        writeln('Ok.');
    end;
end;
```

Saisie d'une adresse

```
procedure lireAdresse(var f: text;var adr: adresse);
begin
    reset(f);
    with adr do
    begin
        write('Nom : ');readln(f,nom);
        write('Prenom : ');readln(f,prenom);
        write('Adresse : ');readln(f, adresse);
        write('Code postal : ');readln(f, cp);
        write('Ville : ');readln(f, ville);
        write('Telephone : ');readln(f, tel);
        write('E-mail : ');readln(f, mail);
    end;
    close(f);
end;
```

Afficher une adresse

```
procedure afficherAdresse(f:text; adr: adresse);
begin
    rewrite(f);
    with adr do
    begin
        write(f, 'Nom :  ',nom);
        write(f, 'Prenom :  ',prenom);
        write(f, 'Adresse :  ',adresse);
        write(f, 'Code postal :  ',cp);
        write(f, 'Ville :  ',ville);
        write(f, 'Telephone :  ',tel);
        write(f, 'E-mail :  ',mail);
    end;
    close(f);
end;
```

Ajout

```
procedure ajout(var f: carnetAdresse);
var nouveau: adresse;
     choix: integer;
begin
    reset(f);
    seek(f,filesize(f));
    repeat
        saisieAdresse(input, nouveau);
        write(f,nouveau);
        write('Saisir un nouveau correspondant (1) / Fin (0) ? ');
        readln(choix);
    until choix=0;
    close(f);
end;
```

liste

```
procedure liste(var f: carnetAdresse);
var r: adresse;
    res: text;
    choix: integer;
begin
    repeat
        write('Sortie a l''ecran (1) ');
        write(`ou dans un fichier texte (2) ? ');
        readln(choix);
    until (choix=1) or (choix=2);
    if (choix=1) then assigncrt(res)
    else begin
        assign(res,  carnet.txt');
        writeln('Resultat dans le fichier carnet.txt.');
     end;
```

Liste (suite)

```
    reset(f);
    while not eof(f) do
    begin
        read(f,r); afficherAdresse(res, r);
    end;
 close(f);
 end;
```

Accès : saisie de la position d 'accès

```
function saisiepos(var f: carnetAdresse): integer;
var pos: integer;
begin
    reset(f);
    repeat
        write('Position dans le fichier ');
        write('0-',filesize(f)-1,'): ');
        readln(pos);
    until (pos>=0) and (pos<=filesize(f)-1);
    close(f);
    saisiepos:=pos;
end;
```

Accès

```
procedure acces(var f: carnetAdresse);
var pos: integer;
    r: adresse;
begin
    pos:=saisiepos(f);
    reset(f); seek(f,pos);
    read(f,r);
    afficherAdresse(output, r);
    close(f);
end;
```

Modif

```
procedure modif(var f: carnetAdresse);
var pos: integer;
    r: adresse;
begin
    pos:=saisiepos(f);
    saisieAdresse(input,r);
    reset(f);
    seek(f,pos);
    write(f,r);
    close(f);
end;
```

Suppression

```
procedure suppression(var f: carnetAdresse);
var pos, i: integer;
    r: adresse;
    temp: fichierAdresse;
begin
    pos:=saisiepos(f);
    assign(temp, 'temp.fic');
    rewrite(temp);
    reset(f);
    for i:=0 to pos-1 do
    begin
        seek(f,i);
        read(f,r);
        write(temp,r);
    end;
```

Suppression (suite)

```
for i:=pos+1 to filesize(f)-1 do
begin
        seek(f,i);
        read(f,r);
        write(temp,r);
end;
close(f);close(temp);
erase(f);
rename(temp,  'carnet.fic');
assign(f,  'carnet.fic');
writeln;writeln('Ok.');
end;
```

Recherche nom

```
procedure recherche_nom(var f: carnetAdresse);
var nc, pc: string;
      r: adresse;
     trouve: boolean;
begin
    write('Nom a chercher : ');readln(nc);
    write('Prenom a chercher : ');readln(pc);
    trouve:=false; reset(f);
    while (not eof(f)) and (not trouve) do
    begin
            read(f,r);
            trouve:= (r.nom=nc) and (r.prenom=pc);
    end;
    close(f);
```

Recherche nom (suite)

```
    if (trouve)
    then afficherAdresse(output, r)
    else writeln('Personne n 'est pas dans le fichier...');
end;
```

Recherche Ville

```
procedure recherche_ville(var f: carnetAdresse);
var v: string; r: adresse; c: integer;
begin
    write('Ville : ');readln(v);
    reset(f); c:=0;
    while not eof(f) do
    begin
            read(f,r);
            if r.ville=v then
            begin
                    writeln(filepos(f),') ',r.nom,' ',r.prenom);
                    c:=c+1;
            end;
    end;
    close(f);
    if (c=0)then writeln('Personne dans cette ville...');
end;
```

Tri

```
procedure tri(var f: carnetAdresse);
    var r, rMin,aux: adresse; indMin, i, j: integer;
begin
    reset(f);
    for i:=0 to filesize(f)-2 do
    begin
        seek(f,i); read(f,rMin);inMin:=i;
        for j:=i+1 to filesize(f)-1 do
        begin
                seek(f,j);read(f,r);
                if
    (rMin.nom+rMin.prenom)>(r.nom+r.prenom) then
                begin
                        inMin:=j; rMin:=r;
                end;
        end;
        if(indMin <>i) then begin
                        seek(f,i); read(f,aux); seek(f,i);
    write(f,rMin);
                        seek(f,indMin); write(f,aux);
                        end;
    end;
    writeln;writeln('Ok.');
end;
```

Recherche Dichotomique

```
procedure dicho(var f: carnetAdresse);
var nc, pc: string;
    r: adresse;
    trouve: boolean;
    min, max, pos, it: integer;
begin
    write('Nom a chercher : ');readln(nc);
    write('Prenom a chercher : ');readln(pc);
    reset(f);
    trouve:=false;
    it:=0; min:=0;
    max:=filesize(f)-1;
```

Recherche Dichotomique (suite)

```
    while (min<=max) and (not trouve) do
    begin
        pos:=(min+max) div 2;
        seek(f,pos);
        read(f,r);
        it:=it+1;
        if (r.nom+r.prenom)=(nc+pc) then
                trouve:=true
        else if (r.nom+r.prenom)>(nc+pc) then
                max:=pos-1
            else min:=pos+1;
    end;
    close(f);
```

Recherche Dichotomique (suite)

```
{Affichage résultat}
if (trouve) then
begin
        afficher(output, r);
        write('(trouve en ',it,' iteration');
        if it>1 then write('s');
        writeln;
    end;
end
else writeln('Personne ne correspond à ce nom et ce prenom...');
end;
```

Carnet d'adresse: utilisation d'un fichier d'index

- **Création d'un fichier d'index**

```
TYPEEnregistrement}
        {Fichier d 'index}
        cle =   record
                        nom, prenom : string;
                end;
        index = record
                        cl: cle;
                        position: integer;
                 end;
        fichierIndex =file of index;
        {Fichier d 'adresse} Déclaration des types}
        adresse=record
                        cl: cle;
                        adresse, cp, ville,
                        tel, mail: string;
                end;
{Fichier d'enregistrements}
        carnetAdresse =file of adresse;
```

Programme principal

```
program fichiers(iput,output);
var f:carnetAdresse;
   ind: fichierIndex; {triée par ordre croissant des noms et
   prénoms}
...
begin
   assign(f, 'carnet.fic'); assign(ind, 'carnet.index ');
   repeat
        choix := menu;
        case choix of
                1: creation(f,ind);
                2: ajout(f, ind);
                3: liste(f, ind);
                4: acces(f, ind);
                5: modif(f, ind);
                6: suppression(f, ind);
                7: recherche_nom(f, ind);
                8: recherche_ville(f, ind);
        end;
   until choix=0;
end.
```

Création

```
procedure creation(var f: carnetAdresse; var ind:fichierIndex);
var choix: integer;
begin
   repeat
     write('Detruire le carnet existant et son index(1) ou annuler (2) ? ');
     readln(choix);
   until (choix=1) or (choix=2);
   if choix=1 then
   begin
        rewrite(f); rewrite(ind);
        close(f); close(ind);
        writeln('Ok.');
   end;
end;
```

Ajout

```
procedure ajout(var f: carnetAdresse; var ind:fichierIndex);
var nouveau: adresse; in:index;
   choix: integer;
begin
   reset(f); reset(ind);
   seek(f,filesize(f)); seek(ind, filesize(ind));
   repeat
        saisieAdresse(input, nouveau);
        write(f,nouveau);
        in.position:=filepos(ind); { in.position:= filepos(f)-1;}
        in.cl:=nouveau.cl;
        write (ind, in);
        write('Saisir un nouveau correspondant (1) / Fin (0) ? ');
        readln(choix);
   until choix=0;
   close(f); close(ind); trier(ind);
end;
```

liste

```
procedure liste(var f: carnetAdresse; var ind:fichierIndex);
var r : adresse;in:index;
      res: text;
      choix: integer;
begin
    repeat
           write('Sortie a l''ecran (1) ');
           write(`ou dans un fichier texte (2) ? ');
           readln(choix);
    until (choix=1) or (choix=2);
    if (choix=1) then assigncrt(res)
    else begin
           assign(res,  carnet.txt');
           writeln('Resultat dans le fichier carnet.txt.');
      end;
```

Liste (suite)

```
    reset(f); reset(ind);
    while not eof(ind) do
    begin
           read(ind,in); seek(f,in.position); read(f,r);
           writeln(res,'Numero : ',in.position);
           afficherAdresse(res, r );
    end;
close(f);close(res);close(ind);
end;

procedure afficherAdresse(f:text; r :adresse);
begin
      rewrite(f);
      with r do
      begin
            write(f,'Nom :   ',cl.nom);
            write(f,'Prénom :   ',cl.prenom);
            write(f,'Adresse :   ',adresse);
            write(f,'Code postal :   ',cp);
            write(f,'Ville :   ',ville);
            write(f,'Téléphone :   ',tel);
            write(f,'E-mail :   ',mail);
      end;
end;
```

Accès : saisie de la clé d 'accès

```
procedure saisieCle(var c:cle);
var pos: integer;
begin
    writeln(` entrez la clé d 'accès : ` );
    write(` Nom ? ` ); readln(c.nom);
    write(` Prénom ? ` ); readln(c.prenom);
end;
```

Accès : rechercher position d 'accès

```
Function  rechercherPosIndex(c:cle; var ind:fichierIndex):integer;
var pos: integer;
begin
        var in: cle; trouve: boolean;
        min, max, pos, it: integer;
begin
      reset(ind);
      trouve:=false;      it:=0; min:=0;max:=filesize(ind)-1;
      while (min<=max) and (not trouve) do
      begin
            pos:=(min+max) div 2;
            seek(ind,pos);read(ind,in); it:=it+1;
            if (in.cl.nom+in.cl.prenom)=(c.nom+c.prenom)
            then trouve:=true
            else if (in.cl.nom+in.cl.prenom)>(c.nom+c.prenom)
                  then max:=pos-1
                  else min:=pos+1;
      end;
      close(ind);
      if (trouve) then rechercherPos:=pos else rechercherPos:= -1;
end;
```

Accès

```
procedure acces(var f: carnetAdresse; var ind:fichierIndex);
var pos: integer;
      r: adresse; c: cle; in :index;
begin
     saisieCle( c );
    pos:=rechercherPosIndex(c, ind);
     if (pos<>-1)
    then begin
              reset(ind, pos); read(ind,in);reset(f); seek(f,in.position);
              read(f,r); afficherAdresse(output, r );

         end
     else writeln(` non trouvé ');
     close(f);
end;
```

Modif

```
procedure modif(var f:carnetAdresse;var ind:fichierIndex);
var pos: integer;
      r: adresse;c :cle;
begin
    saisieCle(c);
    pos:= rechercherPos(c,ind);
    writeln(` Entrez la nouvelle adresse : ');
    saisieAdress(input,r);
    reset(f);
    seek(f,pos);
    write(f,r);
    close(f);
    trierIndex(ind);
end;
```

Suppression

```
procedure suppression(var f: carnetAdresse; var ind:fichierIndex);
var posIndex, i: integer;
      r: adresse;in:index; c : cle;
      tempAdr: fichierAdresse; tempInd:fichierIndex;
begin
    saisieCle( c); posIndex:=rechercherPosIndex(c,ind);
    reset(ind); seek(ind,posIndex); read(ind,in);
    {suppression dans le fichier d 'adresse}
    assign(tempAdr, 'tempAdr.fic');
    rewrite(tempAdr);
    reset(f);
    for i:=0 to in.position-1 do
    begin
          seek(f,i); read(f,r);write(temp,r);
    end;
```

Suppression (suite)

```
for i:=in.position+1 to filesize(f)-1 do
begin
       seek(f,i);read(f,r);write(tempAdr,r);
end;
close(f);close(tempAdr);
erase(f);
rename(tempAdr,  'carnet.fic');
assign(f,  'carnet.fic');
{suppression dans le fichier index}
```

Suppression (suite)

```
      assign(tempInd, 'tempInd.fic');
      rewrite(tempInd);
      reset(ind);
      for i:=0 to posIndex-1 do
      begin
            seek(ind,i); read(ind,in);write(tempInd,in);
      end;
      for i:=posInd+1 to filesize(ind)-1 do
      begin
            seek(ind,i);read(ind,in);write(tempInd,in);
      end;
      close(ind);close(tempInd);
      erase(ind);
      rename(tempInd,  'carnet.index');
      assign(ind,  'carnet.index');
      writeln('Ok.');
end;
```

Recherche nom

```
procedure recherche_nom(var f: carnetAdresse;var ind:fichierIndex);
var r: adresse;c:cle; in:index;
    trouve: boolean;
begin
    saisieCle( c); posIndex:=rechercherPosIndex(c,ind);
    if (posIndex<>-1)
    then begin
            reset(ind); seek(ind, posIndex);read(ind,in);close(ind);
            reset(f); seek(f,in.position); read(f,r); close(f);
            afficherAdresse(output,r);
         end
    else writeln(' non trouve ');
end;
```

Recherche Ville

```
procedure recherche_ville(var f: carnetAdresse);
var v: string; r: adresse; c: integer;
begin
    Définir un autre fichier d 'index avec comme clé d 'accès
    le nom de la ville ….
end;
```

Tri

```
procedure trierIndex(var ind: fichierIndex);
    var inMin, in, aux: index; posMin, i, j: integer;
begin
    reset(ind);
    for i:=0 to filesize(ind)-2 do
    begin
        seek(ind,i); read(ind,inMin); posMin:=i;
        for j:=i+1 to filesize(ind)-1 do
        begin
          seek(ind,j);read(ind,in);
          if (inMin.cl.nom+inMin.cl.prenom)>(in.cl.nom+in.cl.prenom)
    then
            begin
                 inMin:=in;indMin:= j;
            end;
        end;
        if (posMin<>i) then begin
         seek(ind,i); read(ind, aux); write(ind, inMin);

         seek(ind, posMin);write(ind,aux);
        end;
    end;
    close(ind); writeln;writeln('Ok.');
end;
```